



Man Pages (3), (3C), (3K),
(3N), (3R), (3X), (3X11TSOL):
Library Functions

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 805-8069
November 1999

Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, Trusted Solaris, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software – Government Users Subject to Standard License Terms and Conditions

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, Californie 94303-4900 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Contents

PREFACE 91

Intro(3) 97

accept(3N) 120

adornfc(3) 122

auditwrite(3) 124

au_preselect(3) 137

auth_to_str(3) 139

str_to_auth(3) 139

auth_set_to_str(3) 139

str_to_auth_set(3) 139

free_auth_set(3) 139

get_auth_text(3) 139

chkauth(3) 139

auth_to_str(3) 141

str_to_auth(3) 141

auth_set_to_str(3) 141

str_to_auth_set(3) 141

free_auth_set(3) 141

get_auth_text(3) 141

chkauth(3) 141
au_user_mask(3) 143
aw_strerror(3) 145
aw_errno(3) 145
aw_perror(3) 145
aw_strerror(3) 146
aw_errno(3) 146
aw_perror(3) 146
aw_strerror(3) 147
aw_errno(3) 147
aw_perror(3) 147
blmanifest(3) 148
bcllow(3) 148
bclhigh(3) 148
bsllow(3) 148
bslhigh(3) 148
billo(3) 148
bilhigh(3) 148
bclearlow(3) 148
bclearhigh(3) 148
bclundef(3) 148
bslundef(3) 148
bilundef(3) 148
bclearundef(3) 148
blmanifest(3) 150
bcllow(3) 150
bclhigh(3) 150
bsllow(3) 150

bslhigh(3) 150
bllow(3) 150
bilhigh(3) 150
bclearlow(3) 150
bclearhigh(3) 150
bclundef(3) 150
bslundef(3) 150
bilundef(3) 150
bclearundef(3) 150
btohex(3) 152
bcltoh(3) 152
bsltoh(3) 152
biltoh(3) 152
bcleartoh(3) 152
bcltoh_r(3) 152
bsltoh_r(3) 152
biltoh_r(3) 152
bcleartoh_r(3) 152
h_alloc(3) 152
h_free(3) 152
btohex(3) 155
bcltoh(3) 155
bsltoh(3) 155
biltoh(3) 155
bcleartoh(3) 155
bcltoh_r(3) 155
bsltoh_r(3) 155
biltoh_r(3) 155

bcleartoh_r(3) 155
h_alloc(3) 155
h_free(3) 155
bltos(3) 158
bcltos(3) 158
bsltos(3) 158
biltos(3) 158
bcleartos(3) 158
blmanifest(3) 162
bcflow(3) 162
bclhigh(3) 162
bsflow(3) 162
bslhigh(3) 162
bflow(3) 162
bilhigh(3) 162
bclearlow(3) 162
bclearhigh(3) 162
bclundef(3) 162
bslundef(3) 162
bilundef(3) 162
bclearundef(3) 162
blvalid(3) 164
bslvalid(3) 164
bilvalid(3) 164
bclearvalid(3) 164
blmanifest(3) 166
bcflow(3) 166
bclhigh(3) 166

bsllow(3) 166
bslhigh(3) 166
bllow(3) 166
bilhigh(3) 166
bclearlow(3) 166
bclearhigh(3) 166
bclundef(3) 166
bslundef(3) 166
bilundef(3) 166
bclearundef(3) 166
blmanifest(3) 168
bcllow(3) 168
bclhigh(3) 168
bsllow(3) 168
bslhigh(3) 168
bllow(3) 168
bilhigh(3) 168
bclearlow(3) 168
bclearhigh(3) 168
bclundef(3) 168
bslundef(3) 168
bilundef(3) 168
bclearundef(3) 168
bcltobanner(3) 170
btohex(3) 174
bcltoh(3) 174
bsltoh(3) 174
bilton(3) 174

bcleartoh(3) 174
bcltoh_r(3) 174
bsltoh_r(3) 174
bilstoh_r(3) 174
bcleartoh_r(3) 174
h_alloc(3) 174
h_free(3) 174
btohex(3) 177
bcltoh(3) 177
bsltoh(3) 177
bilstoh(3) 177
bcleartoh(3) 177
bcltoh_r(3) 177
bsltoh_r(3) 177
bilstoh_r(3) 177
bcleartoh_r(3) 177
h_alloc(3) 177
h_free(3) 177
blportion(3) 180
bcltosl(3) 180
bcltoil(3) 180
biltolev(3) 180
getcsl(3) 180
getcil(3) 180
setcsl(3) 180
setcil(3) 180
bltos(3) 182
bcltos(3) 182

bsltos(3) 182
biltos(3) 182
bcleartos(3) 182
blportion(3) 186
bcltosl(3) 186
bcltoil(3) 186
biltolev(3) 186
getcsl(3) 186
getcil(3) 186
setcsl(3) 186
setcil(3) 186
blmanifest(3) 188
bcllow(3) 188
bclhigh(3) 188
bsllow(3) 188
bslhigh(3) 188
billow(3) 188
bilhigh(3) 188
bclearlow(3) 188
bclearhigh(3) 188
bclundef(3) 188
bslundef(3) 188
bilundef(3) 188
bclearundef(3) 188
bilconjoin(3) 190
blcompare(3) 191
blequal(3) 191
bilequal(3) 191

bimequal(3) 191
bldominates(3) 191
blstrictdom(3) 191
bildominates(3) 191
bimdominates(3) 191
blinrange(3) 191
blcompare(3) 193
blequal(3) 193
bilequal(3) 193
bimequal(3) 193
bldominates(3) 193
blstrictdom(3) 193
bildominates(3) 193
bimdominates(3) 193
blinrange(3) 193
blmanifest(3) 195
bcllow(3) 195
bclhigh(3) 195
bsllow(3) 195
bslhigh(3) 195
billow(3) 195
bilhigh(3) 195
bclearlow(3) 195
bclearhigh(3) 195
bclundef(3) 195
bslundef(3) 195
bilundef(3) 195
bclearundef(3) 195

blmanifest(3) 197
bcllow(3) 197
bclhigh(3) 197
bsllow(3) 197
bslhigh(3) 197
billo(3) 197
bilhigh(3) 197
bclearlow(3) 197
bclearhigh(3) 197
bclundef(3) 197
bslundef(3) 197
bilundef(3) 197
bclearundef(3) 197
btohex(3) 199
bcltoh(3) 199
bsltoh(3) 199
bilton(3) 199
bcleartoh(3) 199
bcltoh_r(3) 199
bsltoh_r(3) 199
bilton_r(3) 199
bcleartoh_r(3) 199
h_alloc(3) 199
h_free(3) 199
btohex(3) 202
bcltoh(3) 202
bsltoh(3) 202
bilton(3) 202

bcleartoh(3) 202
bcltoh_r(3) 202
bsltoh_r(3) 202
bilstoh_r(3) 202
bcleartoh_r(3) 202
h_alloc(3) 202
h_free(3) 202
blportion(3) 205
bcltosl(3) 205
bcltoil(3) 205
bilstolev(3) 205
getcsl(3) 205
getcil(3) 205
setcsl(3) 205
setcil(3) 205
bltos(3) 207
bcltos(3) 207
bsltos(3) 207
biltos(3) 207
bcleartos(3) 207
blmanifest(3) 211
bcflow(3) 211
bclhigh(3) 211
bsflow(3) 211
bslhigh(3) 211
billo(3) 211
bilhigh(3) 211
bclearlow(3) 211

bclearhigh(3) 211
 bclundef(3) 211
 bslundef(3) 211
 bilundef(3) 211
 bclearundef(3) 211
 blvalid(3) 213
 bslvalid(3) 213
 bilvalid(3) 213
 bclearvalid(3) 213
 blcompare(3) 215
 blequal(3) 215
 bilequal(3) 215
 bimequal(3) 215
 bldominates(3) 215
 blstrictdom(3) 215
 bildominates(3) 215
 bimdominates(3) 215
 blinrange(3) 215
 blcompare(3) 217
 blequal(3) 217
 bilequal(3) 217
 bimequal(3) 217
 bldominates(3) 217
 blstrictdom(3) 217
 bildominates(3) 217
 bimdominates(3) 217
 blinrange(3) 217
 bind(3N) 219

blcompare(3) 221
 blequal(3) 221
 bilequal(3) 221
 bimequal(3) 221
 bldominates(3) 221
 blstrictdom(3) 221
 bildominates(3) 221
 bimdominates(3) 221
 blinrange(3) 221
 blcompare(3) 223
 blequal(3) 223
 bilequal(3) 223
 bimequal(3) 223
 bldominates(3) 223
 blstrictdom(3) 223
 bildominates(3) 223
 bimdominates(3) 223
 blinrange(3) 223
 blcompare(3) 225
 blequal(3) 225
 bilequal(3) 225
 bimequal(3) 225
 bldominates(3) 225
 blstrictdom(3) 225
 bildominates(3) 225
 bimdominates(3) 225
 blinrange(3) 225
 blcompare(3) 227

blequal(3) 227
bilequal(3) 227
bimequal(3) 227
bldominates(3) 227
blstrictdom(3) 227
bildominates(3) 227
bimdominates(3) 227
blinrange(3) 227
blinset(3) 229
blmanifest(3) 231
bcllow(3) 231
bclhigh(3) 231
bsllow(3) 231
bslhigh(3) 231
bllow(3) 231
bilhigh(3) 231
bclearlow(3) 231
bclearhigh(3) 231
bclundef(3) 231
bslundef(3) 231
bilundef(3) 231
bclearundef(3) 231
blminmax(3) 233
blmaximum(3) 233
blminimum(3) 233
blminmax(3) 234
blmaximum(3) 234
blminimum(3) 234

blminmax(3) 235
blmaximum(3) 235
blminimum(3) 235
blportion(3) 236
bcltosl(3) 236
bcltoil(3) 236
biltolev(3) 236
getcsl(3) 236
getcil(3) 236
setcsl(3) 236
setcil(3) 236
blcompare(3) 238
blequal(3) 238
bilequal(3) 238
bimequal(3) 238
bldominates(3) 238
blstrictdom(3) 238
bildominates(3) 238
bimdominates(3) 238
blinrange(3) 238
bltcolor(3) 240
bltcolor_r(3) 240
bltcolor(3) 242
bltcolor_r(3) 242
bltos(3) 244
bcltos(3) 244
bsltos(3) 244
biltos(3) 244

bcleartos(3) 244
bltype(3) 248
setbltype(3) 248
blvalid(3) 250
bslvalid(3) 250
bilvalid(3) 250
bclearvalid(3) 250
blmanifest(3) 252
bcllow(3) 252
bclhigh(3) 252
bsllow(3) 252
bslhigh(3) 252
billo(3) 252
bilhigh(3) 252
bclearlow(3) 252
bclearhigh(3) 252
bclundef(3) 252
bslundef(3) 252
bilundef(3) 252
bclearundef(3) 252
blmanifest(3) 254
bcllow(3) 254
bclhigh(3) 254
bsllow(3) 254
bslhigh(3) 254
billo(3) 254
bilhigh(3) 254
bclearlow(3) 254

bclearhigh(3) 254
 bclundef(3) 254
 bsundef(3) 254
 bilundef(3) 254
 bclearundef(3) 254
 btohex(3) 256
 bcltoh(3) 256
 bslttoh(3) 256
 bilttoh(3) 256
 bcleartoh(3) 256
 bcltoh_r(3) 256
 bslttoh_r(3) 256
 bilttoh_r(3) 256
 bcleartoh_r(3) 256
 h_alloc(3) 256
 h_free(3) 256
 btohex(3) 259
 bcltoh(3) 259
 bslttoh(3) 259
 bilttoh(3) 259
 bcleartoh(3) 259
 bcltoh_r(3) 259
 bslttoh_r(3) 259
 bilttoh_r(3) 259
 bcleartoh_r(3) 259
 h_alloc(3) 259
 h_free(3) 259
 bltos(3) 262

bcltos(3) 262
bsltos(3) 262
biltos(3) 262
bcleartos(3) 262
blmanifest(3) 266
bcllow(3) 266
bclhigh(3) 266
bsllow(3) 266
bslhigh(3) 266
bllow(3) 266
bilhigh(3) 266
bclearlow(3) 266
bclearhigh(3) 266
bclundef(3) 266
bslundef(3) 266
bilundef(3) 266
bclearundef(3) 266
blvalid(3) 268
bslvalid(3) 268
bilvalid(3) 268
bclearvalid(3) 268
btohex(3) 270
bcltoh(3) 270
bsltoh(3) 270
bilstoh(3) 270
bcleartoh(3) 270
bcltoh_r(3) 270
bsltoh_r(3) 270

biltoh_r(3) 270
bcleartoh_r(3) 270
h_alloc(3) 270
h_free(3) 270
auth_to_str(3) 273
str_to_auth(3) 273
auth_set_to_str(3) 273
str_to_auth_set(3) 273
free_auth_set(3) 273
get_auth_text(3) 273
chkauth(3) 273
rpc_clnt_calls(3N) 275
clnt_call(3N) 275
clnt_freeres(3N) 275
clnt_geterr(3N) 275
clnt_perrno(3N) 275
clnt_perror(3N) 275
clnt_sperrno(3N) 275
clnt_sperror(3N) 275
rpc_broadcast(3N) 275
rpc_broadcast_exp(3N) 275
rpc_call(3N) 275
rpc_clnt_create(3N) 279
clnt_control(3N) 279
clnt_create(3N) 279
clnt_create_timed(3N) 279
clnt_create_vers(3N) 279
clnt_create_vers_timed(3N) 279

clnt_destroy(3N) 279
 clnt_dg_create(3N) 279
 clnt_pcreateerror(3N) 279
 clnt_raw_create(3N) 279
 clnt_spccreateerror(3N) 279
 clnt_tli_create(3N) 279
 clnt_tp_create(3N) 279
 clnt_tp_create_timed(3N) 279
 clnt_vc_create(3N) 279
 rpc_createerr(3N) 279
 rpc_clnt_create(3N) 285
 clnt_control(3N) 285
 clnt_create(3N) 285
 clnt_create_timed(3N) 285
 clnt_create_vers(3N) 285
 clnt_create_vers_timed(3N) 285
 clnt_destroy(3N) 285
 clnt_dg_create(3N) 285
 clnt_pcreateerror(3N) 285
 clnt_raw_create(3N) 285
 clnt_spccreateerror(3N) 285
 clnt_tli_create(3N) 285
 clnt_tp_create(3N) 285
 clnt_tp_create_timed(3N) 285
 clnt_vc_create(3N) 285
 rpc_createerr(3N) 285
 rpc_clnt_create(3N) 291
 clnt_control(3N) 291

clnt_create(3N) 291
clnt_create_timed(3N) 291
clnt_create_vers(3N) 291
clnt_create_vers_timed(3N) 291
clnt_destroy(3N) 291
clnt_dg_create(3N) 291
clnt_pcreateerror(3N) 291
clnt_raw_create(3N) 291
clnt_spccreateerror(3N) 291
clnt_tli_create(3N) 291
clnt_tp_create(3N) 291
clnt_tp_create_timed(3N) 291
clnt_vc_create(3N) 291
rpc_createerr(3N) 291
rpc_clnt_create(3N) 297
clnt_control(3N) 297
clnt_create(3N) 297
clnt_create_timed(3N) 297
clnt_create_vers(3N) 297
clnt_create_vers_timed(3N) 297
clnt_destroy(3N) 297
clnt_dg_create(3N) 297
clnt_pcreateerror(3N) 297
clnt_raw_create(3N) 297
clnt_spccreateerror(3N) 297
clnt_tli_create(3N) 297
clnt_tp_create(3N) 297
clnt_tp_create_timed(3N) 297

clnt_vc_create(3N) 297
 rpc_createerr(3N) 297
 rpc_clnt_create(3N) 303
 clnt_control(3N) 303
 clnt_create(3N) 303
 clnt_create_timed(3N) 303
 clnt_create_vers(3N) 303
 clnt_create_vers_timed(3N) 303
 clnt_destroy(3N) 303
 clnt_dg_create(3N) 303
 clnt_pcreateerror(3N) 303
 clnt_raw_create(3N) 303
 clnt_spccreateerror(3N) 303
 clnt_tli_create(3N) 303
 clnt_tp_create(3N) 303
 clnt_tp_create_timed(3N) 303
 clnt_vc_create(3N) 303
 rpc_createerr(3N) 303
 rpc_clnt_create(3N) 309
 clnt_control(3N) 309
 clnt_create(3N) 309
 clnt_create_timed(3N) 309
 clnt_create_vers(3N) 309
 clnt_create_vers_timed(3N) 309
 clnt_destroy(3N) 309
 clnt_dg_create(3N) 309
 clnt_pcreateerror(3N) 309
 clnt_raw_create(3N) 309

clnt_spcrcreateerror(3N) 309
 clnt_tli_create(3N) 309
 clnt_tp_create(3N) 309
 clnt_tp_create_timed(3N) 309
 clnt_vc_create(3N) 309
 rpc_createerr(3N) 309
 rpc_clnt_create(3N) 315
 clnt_control(3N) 315
 clnt_create(3N) 315
 clnt_create_timed(3N) 315
 clnt_create_vers(3N) 315
 clnt_create_vers_timed(3N) 315
 clnt_destroy(3N) 315
 clnt_dg_create(3N) 315
 clnt_pcreateerror(3N) 315
 clnt_raw_create(3N) 315
 clnt_spcrcreateerror(3N) 315
 clnt_tli_create(3N) 315
 clnt_tp_create(3N) 315
 clnt_tp_create_timed(3N) 315
 clnt_vc_create(3N) 315
 rpc_createerr(3N) 315
 rpc_clnt_calls(3N) 321
 clnt_call(3N) 321
 clnt_freeres(3N) 321
 clnt_geterr(3N) 321
 clnt_perrno(3N) 321
 clnt_perror(3N) 321

clnt_sperrno(3N) 321
clnt_sperror(3N) 321
rpc_broadcast(3N) 321
rpc_broadcast_exp(3N) 321
rpc_call(3N) 321
rpc_clnt_calls(3N) 325
clnt_call(3N) 325
clnt_freeres(3N) 325
clnt_geterr(3N) 325
clnt_perrno(3N) 325
clnt_perror(3N) 325
clnt_sperrno(3N) 325
clnt_sperror(3N) 325
rpc_broadcast(3N) 325
rpc_broadcast_exp(3N) 325
rpc_call(3N) 325
rpc_clnt_create(3N) 329
clnt_control(3N) 329
clnt_create(3N) 329
clnt_create_timed(3N) 329
clnt_create_vers(3N) 329
clnt_create_vers_timed(3N) 329
clnt_destroy(3N) 329
clnt_dg_create(3N) 329
clnt_pcreateerror(3N) 329
clnt_raw_create(3N) 329
clnt_spcreateerror(3N) 329
clnt_tli_create(3N) 329

clnt_tp_create(3N) 329
 clnt_tp_create_timed(3N) 329
 clnt_vc_create(3N) 329
 rpc_createerr(3N) 329
 rpc_clnt_calls(3N) 335
 clnt_call(3N) 335
 clnt_freeres(3N) 335
 clnt_geterr(3N) 335
 clnt_perrno(3N) 335
 clnt_perror(3N) 335
 clnt_sperrno(3N) 335
 clnt_sperror(3N) 335
 rpc_broadcast(3N) 335
 rpc_broadcast_exp(3N) 335
 rpc_call(3N) 335
 rpc_clnt_calls(3N) 339
 clnt_call(3N) 339
 clnt_freeres(3N) 339
 clnt_geterr(3N) 339
 clnt_perrno(3N) 339
 clnt_perror(3N) 339
 clnt_sperrno(3N) 339
 clnt_sperror(3N) 339
 rpc_broadcast(3N) 339
 rpc_broadcast_exp(3N) 339
 rpc_call(3N) 339
 rpc_clnt_create(3N) 343
 clnt_control(3N) 343

clnt_create(3N) 343
clnt_create_timed(3N) 343
clnt_create_vers(3N) 343
clnt_create_vers_timed(3N) 343
clnt_destroy(3N) 343
clnt_dg_create(3N) 343
clnt_pcreateerror(3N) 343
clnt_raw_create(3N) 343
clnt_spccreateerror(3N) 343
clnt_tli_create(3N) 343
clnt_tp_create(3N) 343
clnt_tp_create_timed(3N) 343
clnt_vc_create(3N) 343
rpc_createerr(3N) 343
rpc_clnt_create(3N) 349
clnt_control(3N) 349
clnt_create(3N) 349
clnt_create_timed(3N) 349
clnt_create_vers(3N) 349
clnt_create_vers_timed(3N) 349
clnt_destroy(3N) 349
clnt_dg_create(3N) 349
clnt_pcreateerror(3N) 349
clnt_raw_create(3N) 349
clnt_spccreateerror(3N) 349
clnt_tli_create(3N) 349
clnt_tp_create(3N) 349
clnt_tp_create_timed(3N) 349

clnt_vc_create(3N) 349
 rpc_createerr(3N) 349
 rpc_clnt_calls(3N) 355
 clnt_call(3N) 355
 clnt_freeres(3N) 355
 clnt_geterr(3N) 355
 clnt_perrno(3N) 355
 clnt_perror(3N) 355
 clnt_sperrno(3N) 355
 clnt_sperror(3N) 355
 rpc_broadcast(3N) 355
 rpc_broadcast_exp(3N) 355
 rpc_call(3N) 355
 rpc_clnt_calls(3N) 359
 clnt_call(3N) 359
 clnt_freeres(3N) 359
 clnt_geterr(3N) 359
 clnt_perrno(3N) 359
 clnt_perror(3N) 359
 clnt_sperrno(3N) 359
 clnt_sperror(3N) 359
 rpc_broadcast(3N) 359
 rpc_broadcast_exp(3N) 359
 rpc_call(3N) 359
 rpc_clnt_create(3N) 363
 clnt_control(3N) 363
 clnt_create(3N) 363
 clnt_create_timed(3N) 363

clnt_create_vers(3N) 363
 clnt_create_vers_timed(3N) 363
 clnt_destroy(3N) 363
 clnt_dg_create(3N) 363
 clnt_pcreateerror(3N) 363
 clnt_raw_create(3N) 363
 clnt_spccreateerror(3N) 363
 clnt_tli_create(3N) 363
 clnt_tp_create(3N) 363
 clnt_tp_create_timed(3N) 363
 clnt_vc_create(3N) 363
 rpc_createerr(3N) 363
 rpc_clnt_create(3N) 369
 clnt_control(3N) 369
 clnt_create(3N) 369
 clnt_create_timed(3N) 369
 clnt_create_vers(3N) 369
 clnt_create_vers_timed(3N) 369
 clnt_destroy(3N) 369
 clnt_dg_create(3N) 369
 clnt_pcreateerror(3N) 369
 clnt_raw_create(3N) 369
 clnt_spccreateerror(3N) 369
 clnt_tli_create(3N) 369
 clnt_tp_create(3N) 369
 clnt_tp_create_timed(3N) 369
 clnt_vc_create(3N) 369
 rpc_createerr(3N) 369

rpc_clnt_create(3N) 375
 clnt_control(3N) 375
 clnt_create(3N) 375
 clnt_create_timed(3N) 375
 clnt_create_vers(3N) 375
 clnt_create_vers_timed(3N) 375
 clnt_destroy(3N) 375
 clnt_dg_create(3N) 375
 clnt_pcreateerror(3N) 375
 clnt_raw_create(3N) 375
 clnt_spccreateerror(3N) 375
 clnt_tli_create(3N) 375
 clnt_tp_create(3N) 375
 clnt_tp_create_timed(3N) 375
 clnt_vc_create(3N) 375
 rpc_createerr(3N) 375
 rpc_clnt_create(3N) 381
 clnt_control(3N) 381
 clnt_create(3N) 381
 clnt_create_timed(3N) 381
 clnt_create_vers(3N) 381
 clnt_create_vers_timed(3N) 381
 clnt_destroy(3N) 381
 clnt_dg_create(3N) 381
 clnt_pcreateerror(3N) 381
 clnt_raw_create(3N) 381
 clnt_spccreateerror(3N) 381
 clnt_tli_create(3N) 381

clnt_tp_create(3N) 381
clnt_tp_create_timed(3N) 381
clnt_vc_create(3N) 381
rpc_createerr(3N) 381
clock_settime(3R) 387
clock_gettime(3R) 387
clock_getres(3R) 387
clock_settime(3R) 389
clock_gettime(3R) 389
clock_getres(3R) 389
clock_settime(3R) 391
clock_gettime(3R) 391
clock_getres(3R) 391
resolver(3N) 393
res_init(3N) 393
res_mkquery(3N) 393
res_mkupdate(3N) 393
res_mkupdrec(3N) 393
res_query(3N) 393
res_search(3N) 393
res_send(3N) 393
res_update(3N) 393
dn_comp(3N) 393
dn_expand(3N) 393
resolver(3N) 399
res_init(3N) 399
res_mkquery(3N) 399
res_mkupdate(3N) 399

res_mkupdrec(3N) 399
res_query(3N) 399
res_search(3N) 399
res_send(3N) 399
res_update(3N) 399
dn_comp(3N) 399
dn_expand(3N) 399
door_create(3X) 405
door_tcred(3X) 407
getacinfo(3) 409
getacdir(3) 409
getacflg(3) 409
getacmin(3) 409
getacna(3) 409
setac(3) 409
endac(3) 409
getauclassent(3) 411
getauclassnam(3) 411
setauclass(3) 411
endauclass(3) 411
getauclassnam_r(3) 411
getauclassent_r(3) 411
getauevent(3) 413
getauevnam(3) 413
getauevnum(3) 413
getauevnonam(3) 413
setauevent(3) 413
endauevent(3) 413

getauevent_r(3) 413
getauevnam_r(3) 413
getauevnum_r(3) 413
getauusernam(3) 416
getauuserent(3) 416
setauuser(3) 416
endauuser(3) 416
getprofent(3) 418
setprofent(3) 418
endprofent(3) 418
getprofentbyname(3) 418
free_profent(3) 418
getprofstr(3) 421
putprofstr(3) 421
setprofstr(3) 421
endprofstr(3) 421
getprofstrbyname(3) 421
free_profstr(3) 421
getuserent(3) 424
setuserent(3) 424
enduserent(3) 424
getuserentbyname(3) 424
getuserentbyuid(3) 424
free_userent(3) 424
getutent(3C) 427
getutid(3C) 427
getutline(3C) 427
pututline(3C) 427

setutent(3C) 427
endutent(3C) 427
utmpname(3C) 427
getutxent(3C) 430
getutxid(3C) 430
getutxline(3C) 430
pututxline(3C) 430
setutxent(3C) 430
endutxent(3C) 430
utmpxname(3C) 430
getutmp(3C) 430
getutmpx(3C) 430
updwtmp(3C) 430
updwtmpx(3C) 430
auth_to_str(3) 434
str_to_auth(3) 434
auth_set_to_str(3) 434
str_to_auth_set(3) 434
free_auth_set(3) 434
get_auth_text(3) 434
chkauth(3) 434
getprofent(3) 436
setprofent(3) 436
endprofent(3) 436
getprofentbyname(3) 436
free_profent(3) 436
getprofstr(3) 439
putprofstr(3) 439

setprofstr(3) 439
endprofstr(3) 439
getprofstrbyname(3) 439
free_profstr(3) 439
getuserent(3) 442
setuserent(3) 442
enduserent(3) 442
getuserentbyname(3) 442
getuserentbyuid(3) 442
free_userent(3) 442
ftw(3C) 445
nftw(3C) 445
getacinfo(3) 449
getacdir(3) 449
getacflg(3) 449
getacmin(3) 449
getacna(3) 449
setac(3) 449
endac(3) 449
getacinfo(3) 451
getacdir(3) 451
getacflg(3) 451
getacmin(3) 451
getacna(3) 451
setac(3) 451
endac(3) 451
getacinfo(3) 453
getacdir(3) 453

getacflg(3) 453
getacmin(3) 453
getacna(3) 453
setac(3) 453
endac(3) 453
getacinfo(3) 455
getacdir(3) 455
getacflg(3) 455
getacmin(3) 455
getacna(3) 455
setac(3) 455
endac(3) 455
getacinfo(3) 457
getacdir(3) 457
getacflg(3) 457
getacmin(3) 457
getacna(3) 457
setac(3) 457
endac(3) 457
getauclassent(3) 459
getauclassnam(3) 459
setauclass(3) 459
endauclass(3) 459
getauclassnam_r(3) 459
getauclassent_r(3) 459
getauclassent(3) 461
getauclassnam(3) 461
setauclass(3) 461

endauclass(3) 461
getauclassnam_r(3) 461
getauclassent_r(3) 461
getauclassent(3) 463
getauclassnam(3) 463
setauclass(3) 463
endauclass(3) 463
getauclassnam_r(3) 463
getauclassent_r(3) 463
getauclassent(3) 465
getauclassnam(3) 465
setauclass(3) 465
endauclass(3) 465
getauclassnam_r(3) 465
getauclassent_r(3) 465
getauditflags(3) 467
getauditflagsbin(3) 467
getauditflagschar(3) 467
getauditflags(3) 469
getauditflagsbin(3) 469
getauditflagschar(3) 469
getauditflags(3) 471
getauditflagsbin(3) 471
getauditflagschar(3) 471
getauevent(3) 473
getauevnam(3) 473
getauevnum(3) 473
getauevnonam(3) 473

setauevent(3) 473
endauevent(3) 473
getauevent_r(3) 473
getauevnam_r(3) 473
getauevnum_r(3) 473
getauevent(3) 476
getauevnam(3) 476
getauevnum(3) 476
getauevnonam(3) 476
setauevent(3) 476
endauevent(3) 476
getauevent_r(3) 476
getauevnam_r(3) 476
getauevnum_r(3) 476
getauevent(3) 479
getauevnam(3) 479
getauevnum(3) 479
getauevnonam(3) 479
setauevent(3) 479
endauevent(3) 479
getauevent_r(3) 479
getauevnam_r(3) 479
getauevnum_r(3) 479
getauevent(3) 482
getauevnam(3) 482
getauevnum(3) 482
getauevnonam(3) 482
setauevent(3) 482

endauevent(3) 482
getauevent_r(3) 482
getauevnam_r(3) 482
getauevnum_r(3) 482
getauevent(3) 485
getauevnam(3) 485
getauevnum(3) 485
getauevnonam(3) 485
setauevent(3) 485
endauevent(3) 485
getauevent_r(3) 485
getauevnam_r(3) 485
getauevnum_r(3) 485
getauevent(3) 488
getauevnam(3) 488
getauevnum(3) 488
getauevnonam(3) 488
setauevent(3) 488
endauevent(3) 488
getauevent_r(3) 488
getauevnam_r(3) 488
getauevnum_r(3) 488
getauevent(3) 491
getauevnam(3) 491
getauevnum(3) 491
getauevnonam(3) 491
setauevent(3) 491
endauevent(3) 491

getauevent_r(3) 491
getauevnam_r(3) 491
getauevnum_r(3) 491
auth_to_str(3) 494
str_to_auth(3) 494
auth_set_to_str(3) 494
str_to_auth_set(3) 494
free_auth_set(3) 494
get_auth_text(3) 494
chkauth(3) 494
getauusernam(3) 496
getauuserent(3) 496
setauuser(3) 496
endauuser(3) 496
getauusernam(3) 498
getauuserent(3) 498
setauuser(3) 498
endauuser(3) 498
blportion(3) 500
bcltosl(3) 500
bcltoil(3) 500
biltolev(3) 500
getcsl(3) 500
getcil(3) 500
setcsl(3) 500
setcil(3) 500
blportion(3) 502
bcltosl(3) 502

bcltoil(3) 502
biltolev(3) 502
getcsl(3) 502
getcil(3) 502
setcsl(3) 502
setcil(3) 502
getfauditflags(3) 504
getpeerinfo(3) 506
priv_to_str(3) 508
priv_set_to_str(3) 508
str_to_priv(3) 508
str_to_priv_set(3) 508
get_priv_text(3) 508
getprofent(3) 511
setprofent(3) 511
endprofent(3) 511
getprofentbyname(3) 511
free_profent(3) 511
getprofent(3) 514
setprofent(3) 514
endprofent(3) 514
getprofentbyname(3) 514
free_profent(3) 514
getprofstr(3) 517
putprofstr(3) 517
setprofstr(3) 517
endprofstr(3) 517
getprofstrbyname(3) 517

free_profstr(3) 517
getprofstr(3) 520
putprofstr(3) 520
setprofstr(3) 520
endprofstr(3) 520
getprofstrbyname(3) 520
free_profstr(3) 520
getsockopt(3N) 523
setsockopt(3N) 523
getuserent(3) 527
setuserent(3) 527
enduserent(3) 527
getuserentbyname(3) 527
getuserentbyuid(3) 527
free_userent(3) 527
getuserent(3) 530
setuserent(3) 530
enduserent(3) 530
getuserentbyname(3) 530
getuserentbyuid(3) 530
free_userent(3) 530
getuserent(3) 533
setuserent(3) 533
enduserent(3) 533
getuserentbyname(3) 533
getuserentbyuid(3) 533
free_userent(3) 533
getutent(3C) 536

getutid(3C) 536
getutline(3C) 536
pututline(3C) 536
setutent(3C) 536
endutent(3C) 536
utmpname(3C) 536
getutent(3C) 539
getutid(3C) 539
getutline(3C) 539
pututline(3C) 539
setutent(3C) 539
endutent(3C) 539
utmpname(3C) 539
getutent(3C) 542
getutid(3C) 542
getutline(3C) 542
pututline(3C) 542
setutent(3C) 542
endutent(3C) 542
utmpname(3C) 542
getutxent(3C) 545
getutxid(3C) 545
getutxline(3C) 545
pututxline(3C) 545
setutxent(3C) 545
endutxent(3C) 545
utmpxname(3C) 545
getutmp(3C) 545

getutmpx(3C) 545
 updwtmp(3C) 545
 updwtmpx(3C) 545
 getutxent(3C) 549
 getutxid(3C) 549
 getutxline(3C) 549
 pututxline(3C) 549
 setutxent(3C) 549
 endutxent(3C) 549
 utmpxname(3C) 549
 getutmp(3C) 549
 getutmpx(3C) 549
 updwtmp(3C) 549
 updwtmpx(3C) 549
 getutxent(3C) 553
 getutxid(3C) 553
 getutxline(3C) 553
 pututxline(3C) 553
 setutxent(3C) 553
 endutxent(3C) 553
 utmpxname(3C) 553
 getutmp(3C) 553
 getutmpx(3C) 553
 updwtmp(3C) 553
 updwtmpx(3C) 553
 getutxent(3C) 557
 getutxid(3C) 557
 getutxline(3C) 557

pututxline(3C) 557
setutxent(3C) 557
endutxent(3C) 557
utmpxname(3C) 557
getutmp(3C) 557
getutmpx(3C) 557
updwtmp(3C) 557
updwtmpx(3C) 557
getutxent(3C) 561
getutxid(3C) 561
getutxline(3C) 561
pututxline(3C) 561
setutxent(3C) 561
endutxent(3C) 561
utmpxname(3C) 561
getutmp(3C) 561
getutmpx(3C) 561
updwtmp(3C) 561
updwtmpx(3C) 561
getvfaent(3) 565
getvfafile(3) 565
getvfaent(3) 567
getvfafile(3) 567
btohex(3) 569
bcltoh(3) 569
bsltoh(3) 569
bilstoh(3) 569
bcleartoh(3) 569

bcltoh_r(3) 569
 bslttoh_r(3) 569
 bilttoh_r(3) 569
 bclearttoh_r(3) 569
 h_alloc(3) 569
 h_free(3) 569
 hextob(3) 572
 htobcl(3) 572
 htobsl(3) 572
 htobil(3) 572
 htobclear(3) 572
 btohex(3) 574
 bcltoh(3) 574
 bslttoh(3) 574
 bilttoh(3) 574
 bclearttoh(3) 574
 bcltoh_r(3) 574
 bslttoh_r(3) 574
 bilttoh_r(3) 574
 bclearttoh_r(3) 574
 h_alloc(3) 574
 h_free(3) 574
 hextob(3) 577
 htobcl(3) 577
 htobsl(3) 577
 htobil(3) 577
 htobclear(3) 577
 hextob(3) 579

htobcl(3) 579
htobsl(3) 579
htobil(3) 579
htobclear(3) 579
hextob(3) 581
htobcl(3) 581
htobsl(3) 581
htobil(3) 581
htobclear(3) 581
hextob(3) 583
htobcl(3) 583
htobsl(3) 583
htobil(3) 583
htobclear(3) 583
initgroups(3C) 585
kstat_read(3K) 586
kstat_write(3K) 586
kstat_read(3K) 587
kstat_write(3K) 587
labelbuilder(3) 588
tsol_lbuild_create(3) 588
tsol_lbuild_get(3) 588
tsol_lbuild_set(3) 588
tsol_lbuild_destroy(3) 588
labelclipping(3) 594
Xbcltos(3) 594
Xbsltos(3) 594
Xbiltos(3) 594

Xbclear(3) 594
labelinfo(3) 596
labelvers(3) 598
libt6(3N) 600
listen(3N) 605
mldgetcwd(3) 607
mldstat(3) 609
mldlstat(3) 609
mldrealpath(3) 611
mldrealpathl(3) 611
mldrealpath(3) 613
mldrealpathl(3) 613
mldstat(3) 615
mldlstat(3) 615
mlock(3C) 617
munlock(3C) 617
mlockall(3C) 619
munlockall(3C) 619
mlock(3C) 621
munlock(3C) 621
mlockall(3C) 623
munlockall(3C) 623
ftw(3C) 625
nftw(3C) 625
nis_names(3N) 629
nis_lookup(3N) 629
nis_add(3N) 629
nis_remove(3N) 629

nis_modify(3N) 629
nis_freeresult(3N) 629
nis_tables(3N) 635
nis_list(3N) 635
nis_add_entry(3N) 635
nis_remove_entry(3N) 635
nis_modify_entry(3N) 635
nis_first_entry(3N) 635
nis_next_entry(3N) 635
nis_groups(3N) 644
nis_ismember(3N) 644
nis_addmember(3N) 644
nis_removemember(3N) 644
nis_creategroup(3N) 644
nis_destroygroup(3N) 644
nis_verifygroup(3N) 644
nis_print_group_entry(3N) 644
nis_ping(3N) 647
nis_checkpoint(3N) 647
nis_groups(3N) 649
nis_ismember(3N) 649
nis_addmember(3N) 649
nis_removemember(3N) 649
nis_creategroup(3N) 649
nis_destroygroup(3N) 649
nis_verifygroup(3N) 649
nis_print_group_entry(3N) 649
nis_groups(3N) 652

nis_ismember(3N) 652
nis_addmember(3N) 652
nis_removemember(3N) 652
nis_creategroup(3N) 652
nis_destroygroup(3N) 652
nis_verifygroup(3N) 652
nis_print_group_entry(3N) 652
nis_tables(3N) 655
nis_list(3N) 655
nis_add_entry(3N) 655
nis_remove_entry(3N) 655
nis_modify_entry(3N) 655
nis_first_entry(3N) 655
nis_next_entry(3N) 655
nis_names(3N) 664
nis_lookup(3N) 664
nis_add(3N) 664
nis_remove(3N) 664
nis_modify(3N) 664
nis_freeresult(3N) 664
nis_server(3N) 670
nis_mkdir(3N) 670
nis_rmdir(3N) 670
nis_servstate(3N) 670
nis_stats(3N) 670
nis_getservlist(3N) 670
nis_freeservlist(3N) 670
nis_freetags(3N) 670

nis_server(3N) 672
nis_mkdir(3N) 672
nis_rmdir(3N) 672
nis_servstate(3N) 672
nis_stats(3N) 672
nis_getservlist(3N) 672
nis_freeservlist(3N) 672
nis_freetags(3N) 672
nis_server(3N) 674
nis_mkdir(3N) 674
nis_rmdir(3N) 674
nis_servstate(3N) 674
nis_stats(3N) 674
nis_getservlist(3N) 674
nis_freeservlist(3N) 674
nis_freetags(3N) 674
nis_groups(3N) 676
nis_ismember(3N) 676
nis_addmember(3N) 676
nis_removemember(3N) 676
nis_creategroup(3N) 676
nis_destroygroup(3N) 676
nis_verifygroup(3N) 676
nis_print_group_entry(3N) 676
nis_groups(3N) 679
nis_ismember(3N) 679
nis_addmember(3N) 679
nis_removemember(3N) 679

nis_creategroup(3N) 679
nis_destroygroup(3N) 679
nis_verifygroup(3N) 679
nis_print_group_entry(3N) 679
nis_tables(3N) 682
nis_list(3N) 682
nis_add_entry(3N) 682
nis_remove_entry(3N) 682
nis_modify_entry(3N) 682
nis_first_entry(3N) 682
nis_next_entry(3N) 682
nis_names(3N) 691
nis_lookup(3N) 691
nis_add(3N) 691
nis_remove(3N) 691
nis_modify(3N) 691
nis_freeresult(3N) 691
nis_server(3N) 697
nis_mkdir(3N) 697
nis_rmdir(3N) 697
nis_servstate(3N) 697
nis_stats(3N) 697
nis_getservlist(3N) 697
nis_freeservlist(3N) 697
nis_freetags(3N) 697
nis_names(3N) 699
nis_lookup(3N) 699
nis_add(3N) 699

nis_remove(3N) 699
nis_modify(3N) 699
nis_freeresult(3N) 699
nis_tables(3N) 705
nis_list(3N) 705
nis_add_entry(3N) 705
nis_remove_entry(3N) 705
nis_modify_entry(3N) 705
nis_first_entry(3N) 705
nis_next_entry(3N) 705
nis_names(3N) 714
nis_lookup(3N) 714
nis_add(3N) 714
nis_remove(3N) 714
nis_modify(3N) 714
nis_freeresult(3N) 714
nis_tables(3N) 720
nis_list(3N) 720
nis_add_entry(3N) 720
nis_remove_entry(3N) 720
nis_modify_entry(3N) 720
nis_first_entry(3N) 720
nis_next_entry(3N) 720
nis_ping(3N) 729
nis_checkpoint(3N) 729
nis_groups(3N) 731
nis_ismember(3N) 731
nis_addmember(3N) 731

nis_removemember(3N) 731
nis_creategroup(3N) 731
nis_destroygroup(3N) 731
nis_verifygroup(3N) 731
nis_print_group_entry(3N) 731
nis_names(3N) 734
nis_lookup(3N) 734
nis_add(3N) 734
nis_remove(3N) 734
nis_modify(3N) 734
nis_freeresult(3N) 734
nis_tables(3N) 740
nis_list(3N) 740
nis_add_entry(3N) 740
nis_remove_entry(3N) 740
nis_modify_entry(3N) 740
nis_first_entry(3N) 740
nis_next_entry(3N) 740
nis_groups(3N) 749
nis_ismember(3N) 749
nis_addmember(3N) 749
nis_removemember(3N) 749
nis_creategroup(3N) 749
nis_destroygroup(3N) 749
nis_verifygroup(3N) 749
nis_print_group_entry(3N) 749
nis_server(3N) 752
nis_mkdir(3N) 752

nis_rmdir(3N) 752
nis_servstate(3N) 752
nis_stats(3N) 752
nis_getservlist(3N) 752
nis_freeservlist(3N) 752
nis_freetags(3N) 752
nis_server(3N) 754
nis_mkdir(3N) 754
nis_rmdir(3N) 754
nis_servstate(3N) 754
nis_stats(3N) 754
nis_getservlist(3N) 754
nis_freeservlist(3N) 754
nis_freetags(3N) 754
nis_server(3N) 756
nis_mkdir(3N) 756
nis_rmdir(3N) 756
nis_servstate(3N) 756
nis_stats(3N) 756
nis_getservlist(3N) 756
nis_freeservlist(3N) 756
nis_freetags(3N) 756
nis_server(3N) 758
nis_mkdir(3N) 758
nis_rmdir(3N) 758
nis_servstate(3N) 758
nis_stats(3N) 758
nis_getservlist(3N) 758

nis_freeservlist(3N) 758
nis_frehtags(3N) 758
nis_tables(3N) 760
nis_list(3N) 760
nis_add_entry(3N) 760
nis_remove_entry(3N) 760
nis_modify_entry(3N) 760
nis_first_entry(3N) 760
nis_next_entry(3N) 760
nis_groups(3N) 769
nis_ismember(3N) 769
nis_addmember(3N) 769
nis_removemember(3N) 769
nis_creategroup(3N) 769
nis_destroygroup(3N) 769
nis_verifygroup(3N) 769
nis_print_group_entry(3N) 769
plock(3) 772
priv_to_str(3) 774
priv_set_to_str(3) 774
str_to_priv(3) 774
str_to_priv_set(3) 774
get_priv_text(3) 774
priv_to_str(3) 777
priv_set_to_str(3) 777
str_to_priv(3) 777
str_to_priv_set(3) 777
get_priv_text(3) 777

getprofstr(3) 780
putprofstr(3) 780
setprofstr(3) 780
endprofstr(3) 780
getprofstrbyname(3) 780
free_profstr(3) 780
getutent(3C) 783
getutid(3C) 783
getutline(3C) 783
pututline(3C) 783
setutent(3C) 783
endutent(3C) 783
utmpname(3C) 783
getutxent(3C) 786
getutxid(3C) 786
getutxline(3C) 786
pututxline(3C) 786
setutxent(3C) 786
endutxent(3C) 786
utmpxname(3C) 786
getutmp(3C) 786
getutmpx(3C) 786
updwtmp(3C) 786
updwtmpx(3C) 786
randomword(3) 790
resolver(3N) 792
res_init(3N) 792
res_mkquery(3N) 792

res_mkupdate(3N) 792
 res_mkupdrec(3N) 792
 res_query(3N) 792
 res_search(3N) 792
 res_send(3N) 792
 res_update(3N) 792
 dn_comp(3N) 792
 dn_expand(3N) 792
 resolver(3N) 798
 res_init(3N) 798
 res_mkquery(3N) 798
 res_mkupdate(3N) 798
 res_mkupdrec(3N) 798
 res_query(3N) 798
 res_search(3N) 798
 res_send(3N) 798
 res_update(3N) 798
 dn_comp(3N) 798
 dn_expand(3N) 798
 resolver(3N) 804
 res_init(3N) 804
 res_mkquery(3N) 804
 res_mkupdate(3N) 804
 res_mkupdrec(3N) 804
 res_query(3N) 804
 res_search(3N) 804
 res_send(3N) 804
 res_update(3N) 804

dn_comp(3N) 804
dn_expand(3N) 804
resolver(3N) 810
res_init(3N) 810
res_mkquery(3N) 810
res_mkupdate(3N) 810
res_mkupdrec(3N) 810
res_query(3N) 810
res_search(3N) 810
res_send(3N) 810
res_update(3N) 810
dn_comp(3N) 810
dn_expand(3N) 810
resolver(3N) 816
res_init(3N) 816
res_mkquery(3N) 816
res_mkupdate(3N) 816
res_mkupdrec(3N) 816
res_query(3N) 816
res_search(3N) 816
res_send(3N) 816
res_update(3N) 816
dn_comp(3N) 816
dn_expand(3N) 816
resolver(3N) 822
res_init(3N) 822
res_mkquery(3N) 822
res_mkupdate(3N) 822

res_mkupdrec(3N) 822
 res_query(3N) 822
 res_search(3N) 822
 res_send(3N) 822
 res_update(3N) 822
 dn_comp(3N) 822
 dn_expand(3N) 822
 resolver(3N) 828
 res_init(3N) 828
 res_mkquery(3N) 828
 res_mkupdate(3N) 828
 res_mkupdrec(3N) 828
 res_query(3N) 828
 res_search(3N) 828
 res_send(3N) 828
 res_update(3N) 828
 dn_comp(3N) 828
 dn_expand(3N) 828
 resolver(3N) 834
 res_init(3N) 834
 res_mkquery(3N) 834
 res_mkupdate(3N) 834
 res_mkupdrec(3N) 834
 res_query(3N) 834
 res_search(3N) 834
 res_send(3N) 834
 res_update(3N) 834
 dn_comp(3N) 834

dn_expand(3N) 834
resolver(3N) 840
res_init(3N) 840
res_mkquery(3N) 840
res_mkupdate(3N) 840
res_mkupdrec(3N) 840
res_query(3N) 840
res_search(3N) 840
res_send(3N) 840
res_update(3N) 840
dn_comp(3N) 840
dn_expand(3N) 840
rpc(3N) 846
rpcbind(3N) 856
rpcb_getmaps(3N) 856
rpcb_getallmaps(3N) 856
rpcb_getaddr(3N) 856
rpcb_gettime(3N) 856
rpcb_rmtcall(3N) 856
rpcb_set(3N) 856
rpcb_unset(3N) 856
rpcbind(3N) 859
rpcb_getmaps(3N) 859
rpcb_getallmaps(3N) 859
rpcb_getaddr(3N) 859
rpcb_gettime(3N) 859
rpcb_rmtcall(3N) 859
rpcb_set(3N) 859

rpcb_unset(3N) 859
rpcbind(3N) 862
rpcb_getmaps(3N) 862
rpcb_getallmaps(3N) 862
rpcb_getaddr(3N) 862
rpcb_gettime(3N) 862
rpcb_rmtcall(3N) 862
rpcb_set(3N) 862
rpcb_unset(3N) 862
rpcbind(3N) 865
rpcb_getmaps(3N) 865
rpcb_getallmaps(3N) 865
rpcb_getaddr(3N) 865
rpcb_gettime(3N) 865
rpcb_rmtcall(3N) 865
rpcb_set(3N) 865
rpcb_unset(3N) 865
rpcbind(3N) 868
rpcb_getmaps(3N) 868
rpcb_getallmaps(3N) 868
rpcb_getaddr(3N) 868
rpcb_gettime(3N) 868
rpcb_rmtcall(3N) 868
rpcb_set(3N) 868
rpcb_unset(3N) 868
rpcbind(3N) 871
rpcb_getmaps(3N) 871
rpcb_getallmaps(3N) 871

rpcb_getaddr(3N) 871
rpcb_gettime(3N) 871
rpcb_rmtcall(3N) 871
rpcb_set(3N) 871
rpcb_unset(3N) 871
rpc_clnt_calls(3N) 874
clnt_call(3N) 874
clnt_freeres(3N) 874
clnt_geterr(3N) 874
clnt_perrno(3N) 874
clnt_perror(3N) 874
clnt_sperrno(3N) 874
clnt_sperror(3N) 874
rpc_broadcast(3N) 874
rpc_broadcast_exp(3N) 874
rpc_call(3N) 874
rpc_clnt_calls(3N) 878
clnt_call(3N) 878
clnt_freeres(3N) 878
clnt_geterr(3N) 878
clnt_perrno(3N) 878
clnt_perror(3N) 878
clnt_sperrno(3N) 878
clnt_sperror(3N) 878
rpc_broadcast(3N) 878
rpc_broadcast_exp(3N) 878
rpc_call(3N) 878
rpcbind(3N) 882

rpcb_getmaps(3N) 882
 rpcb_getallmaps(3N) 882
 rpcb_getaddr(3N) 882
 rpcb_gettime(3N) 882
 rpcb_rmtcall(3N) 882
 rpcb_set(3N) 882
 rpcb_unset(3N) 882
 rpcbind(3N) 885
 rpcb_getmaps(3N) 885
 rpcb_getallmaps(3N) 885
 rpcb_getaddr(3N) 885
 rpcb_gettime(3N) 885
 rpcb_rmtcall(3N) 885
 rpcb_set(3N) 885
 rpcb_unset(3N) 885
 rpc_clnt_calls(3N) 888
 clnt_call(3N) 888
 clnt_freeres(3N) 888
 clnt_geterr(3N) 888
 clnt_perrno(3N) 888
 clnt_perror(3N) 888
 clnt_sperno(3N) 888
 clnt_sperror(3N) 888
 rpc_broadcast(3N) 888
 rpc_broadcast_exp(3N) 888
 rpc_call(3N) 888
 rpc_clnt_calls(3N) 892
 clnt_call(3N) 892

clnt_freeres(3N) 892
clnt_geterr(3N) 892
clnt_perrno(3N) 892
clnt_perror(3N) 892
clnt_sperrno(3N) 892
clnt_sperror(3N) 892
rpc_broadcast(3N) 892
rpc_broadcast_exp(3N) 892
rpc_call(3N) 892
rpc_clnt_create(3N) 896
clnt_control(3N) 896
clnt_create(3N) 896
clnt_create_timed(3N) 896
clnt_create_vers(3N) 896
clnt_create_vers_timed(3N) 896
clnt_destroy(3N) 896
clnt_dg_create(3N) 896
clnt_pcreateerror(3N) 896
clnt_raw_create(3N) 896
clnt_spcreateerror(3N) 896
clnt_tli_create(3N) 896
clnt_tp_create(3N) 896
clnt_tp_create_timed(3N) 896
clnt_vc_create(3N) 896
rpc_createerr(3N) 896
rpc_clnt_create(3N) 902
clnt_control(3N) 902
clnt_create(3N) 902

clnt_create_timed(3N) 902
clnt_create_vers(3N) 902
clnt_create_vers_timed(3N) 902
clnt_destroy(3N) 902
clnt_dg_create(3N) 902
clnt_pcreateerror(3N) 902
clnt_raw_create(3N) 902
clnt_spccreateerror(3N) 902
clnt_tli_create(3N) 902
clnt_tp_create(3N) 902
clnt_tp_create_timed(3N) 902
clnt_vc_create(3N) 902
rpc_createerr(3N) 902
rpc_svc_calls(3N) 908
svc_dg_enablecache(3N) 908
svc_done(3N) 908
svc_exit(3N) 908
svc_fdset(3N) 908
svc_freeargs(3N) 908
svc_getargs(3N) 908
svc_getreq_common(3N) 908
svc_getreq_poll(3N) 908
svc_getreqset(3N) 908
svc_getrpccaller(3N) 908
svc_max_pollfd(3N) 908
svc_pollfd(3N) 908
svc_run(3N) 908
svc_sendreply(3N) 908

rpc_svc_create(3N) 913
svc_control(3N) 913
svc_create(3N) 913
svc_destroy(3N) 913
svc_dg_create(3N) 913
svc_fd_create(3N) 913
svc_raw_create(3N) 913
svc_tli_create(3N) 913
svc_tp_create(3N) 913
svc_vc_create(3N) 913
rpc_svc_reg(3N) 918
rpc_reg(3N) 918
svc_reg(3N) 918
svc_unreg(3N) 918
svc_auth_reg(3N) 918
xpirt_register(3N) 918
xpirt_unregister(3N) 918
sbltos(3) 921
sbcltos(3) 921
sbsltos(3) 921
sbiltos(3) 921
sbcleartos(3) 921
sbltos(3) 924
sbcltos(3) 924
sbsltos(3) 924
sbiltos(3) 924
sbcleartos(3) 924
sbltos(3) 927

sbcltos(3) 927
sbsltos(3) 927
sbiltos(3) 927
sbcleartos(3) 927
sbltos(3) 930
sbcltos(3) 930
sbsltos(3) 930
sbiltos(3) 930
sbcleartos(3) 930
sbltos(3) 933
sbcltos(3) 933
sbsltos(3) 933
sbiltos(3) 933
sbcleartos(3) 933
send(3N) 936
sendto(3N) 936
sendmsg(3N) 936
send(3N) 938
sendto(3N) 938
sendmsg(3N) 938
send(3N) 940
sendto(3N) 940
sendmsg(3N) 940
getacinfo(3) 942
getacdir(3) 942
getacflg(3) 942
getacmin(3) 942
getacna(3) 942

setac(3) 942
endac(3) 942
getaclassent(3) 944
getaclassnam(3) 944
setaclass(3) 944
endaclass(3) 944
getaclassnam_r(3) 944
getaclassent_r(3) 944
getauevent(3) 946
getauevnam(3) 946
getauevnum(3) 946
getauevnonam(3) 946
setauevent(3) 946
endauevent(3) 946
getauevent_r(3) 946
getauevnam_r(3) 946
getauevnum_r(3) 946
getauusernam(3) 949
getauuserent(3) 949
setauuser(3) 949
endauuser(3) 949
bltype(3) 951
setbltype(3) 951
blportion(3) 953
bcltosl(3) 953
bcltoil(3) 953
bilstolev(3) 953
getcsl(3) 953

getcil(3) 953
setcsl(3) 953
setcil(3) 953
blportion(3) 955
bcltosl(3) 955
bcltoil(3) 955
biltolev(3) 955
getcsl(3) 955
getcil(3) 955
setcsl(3) 955
setcil(3) 955
set_effective_priv(3) 957
set_inheritable_priv(3) 957
set_permitted_priv(3) 957
set_effective_priv(3) 959
set_inheritable_priv(3) 959
set_permitted_priv(3) 959
set_effective_priv(3) 961
set_inheritable_priv(3) 961
set_permitted_priv(3) 961
getprofent(3) 963
setprofent(3) 963
endprofent(3) 963
getprofentbyname(3) 963
free_profent(3) 963
getprofstr(3) 966
putprofstr(3) 966
setprofstr(3) 966

endprofstr(3) 966
getprofstrbyname(3) 966
free_profstr(3) 966
getsockopt(3N) 969
setsockopt(3N) 969
getuserent(3) 973
setuserent(3) 973
enduserent(3) 973
getuserentbyname(3) 973
getuserentbyuid(3) 973
free_userent(3) 973
getutent(3C) 976
getutid(3C) 976
getutline(3C) 976
pututline(3C) 976
setutent(3C) 976
endutent(3C) 976
utmpname(3C) 976
getutxent(3C) 979
getutxid(3C) 979
getutxline(3C) 979
pututxline(3C) 979
setutxent(3C) 979
endutxent(3C) 979
utmpxname(3C) 979
getutmp(3C) 979
getutmpx(3C) 979
updwtmp(3C) 979

updwtmpx(3C) 979
stobl(3) 983
stobcl(3) 983
stobsl(3) 983
stobil(3) 983
stobclear(3) 983
stobl(3) 988
stobcl(3) 988
stobsl(3) 988
stobil(3) 988
stobclear(3) 988
stobl(3) 993
stobcl(3) 993
stobsl(3) 993
stobil(3) 993
stobclear(3) 993
stobl(3) 998
stobcl(3) 998
stobsl(3) 998
stobil(3) 998
stobclear(3) 998
stobl(3) 1003
stobcl(3) 1003
stobsl(3) 1003
stobil(3) 1003
stobclear(3) 1003
auth_to_str(3) 1008
str_to_auth(3) 1008

auth_set_to_str(3) 1008
str_to_auth_set(3) 1008
free_auth_set(3) 1008
get_auth_text(3) 1008
chkauth(3) 1008
auth_to_str(3) 1010
str_to_auth(3) 1010
auth_set_to_str(3) 1010
str_to_auth_set(3) 1010
free_auth_set(3) 1010
get_auth_text(3) 1010
chkauth(3) 1010
priv_to_str(3) 1012
priv_set_to_str(3) 1012
str_to_priv(3) 1012
str_to_priv_set(3) 1012
get_priv_text(3) 1012
priv_to_str(3) 1015
priv_set_to_str(3) 1015
str_to_priv(3) 1015
str_to_priv_set(3) 1015
get_priv_text(3) 1015
rpc_svc_reg(3N) 1018
rpc_reg(3N) 1018
svc_reg(3N) 1018
svc_unreg(3N) 1018
svc_auth_reg(3N) 1018
xpirt_register(3N) 1018

xprt_unregister(3N) 1018
rpc_svc_create(3N) 1021
svc_control(3N) 1021
svc_create(3N) 1021
svc_destroy(3N) 1021
svc_dg_create(3N) 1021
svc_fd_create(3N) 1021
svc_raw_create(3N) 1021
svc_tli_create(3N) 1021
svc_tp_create(3N) 1021
svc_vc_create(3N) 1021
rpc_svc_create(3N) 1026
svc_control(3N) 1026
svc_create(3N) 1026
svc_destroy(3N) 1026
svc_dg_create(3N) 1026
svc_fd_create(3N) 1026
svc_raw_create(3N) 1026
svc_tli_create(3N) 1026
svc_tp_create(3N) 1026
svc_vc_create(3N) 1026
rpc_svc_create(3N) 1031
svc_control(3N) 1031
svc_create(3N) 1031
svc_destroy(3N) 1031
svc_dg_create(3N) 1031
svc_fd_create(3N) 1031
svc_raw_create(3N) 1031

svc_tli_create(3N)	1031
svc_tp_create(3N)	1031
svc_vc_create(3N)	1031
rpc_svc_create(3N)	1036
svc_control(3N)	1036
svc_create(3N)	1036
svc_destroy(3N)	1036
svc_dg_create(3N)	1036
svc_fd_create(3N)	1036
svc_raw_create(3N)	1036
svc_tli_create(3N)	1036
svc_tp_create(3N)	1036
svc_vc_create(3N)	1036
rpc_svc_calls(3N)	1041
svc_dg_enablecache(3N)	1041
svc_done(3N)	1041
svc_exit(3N)	1041
svc_fdset(3N)	1041
svc_freeargs(3N)	1041
svc_getargs(3N)	1041
svc_getreq_common(3N)	1041
svc_getreq_poll(3N)	1041
svc_getreqset(3N)	1041
svc_getrpccaller(3N)	1041
svc_max_pollfd(3N)	1041
svc_pollfd(3N)	1041
svc_run(3N)	1041
svc_sendreply(3N)	1041

rpc_svc_calls(3N) 1046
 svc_dg_enablecache(3N) 1046
 svc_done(3N) 1046
 svc_exit(3N) 1046
 svc_fdset(3N) 1046
 svc_freeargs(3N) 1046
 svc_getargs(3N) 1046
 svc_getreq_common(3N) 1046
 svc_getreq_poll(3N) 1046
 svc_getreqset(3N) 1046
 svc_getrpccaller(3N) 1046
 svc_max_pollfd(3N) 1046
 svc_pollfd(3N) 1046
 svc_run(3N) 1046
 svc_sendreply(3N) 1046
 rpc_svc_calls(3N) 1051
 svc_dg_enablecache(3N) 1051
 svc_done(3N) 1051
 svc_exit(3N) 1051
 svc_fdset(3N) 1051
 svc_freeargs(3N) 1051
 svc_getargs(3N) 1051
 svc_getreq_common(3N) 1051
 svc_getreq_poll(3N) 1051
 svc_getreqset(3N) 1051
 svc_getrpccaller(3N) 1051
 svc_max_pollfd(3N) 1051
 svc_pollfd(3N) 1051

svc_run(3N) 1051
svc_sendreply(3N) 1051
rpc_svc_create(3N) 1056
svc_control(3N) 1056
svc_create(3N) 1056
svc_destroy(3N) 1056
svc_dg_create(3N) 1056
svc_fd_create(3N) 1056
svc_raw_create(3N) 1056
svc_tli_create(3N) 1056
svc_tp_create(3N) 1056
svc_vc_create(3N) 1056
rpc_svc_calls(3N) 1061
svc_dg_enablecache(3N) 1061
svc_done(3N) 1061
svc_exit(3N) 1061
svc_fdset(3N) 1061
svc_freeargs(3N) 1061
svc_getargs(3N) 1061
svc_getreq_common(3N) 1061
svc_getreq_poll(3N) 1061
svc_getreqset(3N) 1061
svc_getrpccaller(3N) 1061
svc_max_pollfd(3N) 1061
svc_pollfd(3N) 1061
svc_run(3N) 1061
svc_sendreply(3N) 1061
rpc_svc_calls(3N) 1066

svc_dg_enablecache(3N) 1066
 svc_done(3N) 1066
 svc_exit(3N) 1066
 svc_fdset(3N) 1066
 svc_freeargs(3N) 1066
 svc_getargs(3N) 1066
 svc_getreq_common(3N) 1066
 svc_getreq_poll(3N) 1066
 svc_getreqset(3N) 1066
 svc_getrpccaller(3N) 1066
 svc_max_pollfd(3N) 1066
 svc_pollfd(3N) 1066
 svc_run(3N) 1066
 svc_sendreply(3N) 1066
 rpc_svc_calls(3N) 1071
 svc_dg_enablecache(3N) 1071
 svc_done(3N) 1071
 svc_exit(3N) 1071
 svc_fdset(3N) 1071
 svc_freeargs(3N) 1071
 svc_getargs(3N) 1071
 svc_getreq_common(3N) 1071
 svc_getreq_poll(3N) 1071
 svc_getreqset(3N) 1071
 svc_getrpccaller(3N) 1071
 svc_max_pollfd(3N) 1071
 svc_pollfd(3N) 1071
 svc_run(3N) 1071

svc_sendreply(3N) 1071
 rpc_svc_calls(3N) 1076
 svc_dg_enablecache(3N) 1076
 svc_done(3N) 1076
 svc_exit(3N) 1076
 svc_fdset(3N) 1076
 svc_freeargs(3N) 1076
 svc_getargs(3N) 1076
 svc_getreq_common(3N) 1076
 svc_getreq_poll(3N) 1076
 svc_getreqset(3N) 1076
 svc_getrpccaller(3N) 1076
 svc_max_pollfd(3N) 1076
 svc_pollfd(3N) 1076
 svc_run(3N) 1076
 svc_sendreply(3N) 1076
 rpc_svc_calls(3N) 1081
 svc_dg_enablecache(3N) 1081
 svc_done(3N) 1081
 svc_exit(3N) 1081
 svc_fdset(3N) 1081
 svc_freeargs(3N) 1081
 svc_getargs(3N) 1081
 svc_getreq_common(3N) 1081
 svc_getreq_poll(3N) 1081
 svc_getreqset(3N) 1081
 svc_getrpccaller(3N) 1081
 svc_max_pollfd(3N) 1081

svc_pollfd(3N) 1081
 svc_run(3N) 1081
 svc_sendreply(3N) 1081
 rpc_svc_calls(3N) 1086
 svc_dg_enablecache(3N) 1086
 svc_done(3N) 1086
 svc_exit(3N) 1086
 svc_fdset(3N) 1086
 svc_freeargs(3N) 1086
 svc_getargs(3N) 1086
 svc_getreq_common(3N) 1086
 svc_getreq_poll(3N) 1086
 svc_getreqset(3N) 1086
 svc_getrpccaller(3N) 1086
 svc_max_pollfd(3N) 1086
 svc_pollfd(3N) 1086
 svc_run(3N) 1086
 svc_sendreply(3N) 1086
 rpc_svc_calls(3N) 1091
 svc_dg_enablecache(3N) 1091
 svc_done(3N) 1091
 svc_exit(3N) 1091
 svc_fdset(3N) 1091
 svc_freeargs(3N) 1091
 svc_getargs(3N) 1091
 svc_getreq_common(3N) 1091
 svc_getreq_poll(3N) 1091
 svc_getreqset(3N) 1091

svc_getrpccaller(3N) 1091
svc_max_pollfd(3N) 1091
svc_pollfd(3N) 1091
svc_run(3N) 1091
svc_sendreply(3N) 1091
rpc_svc_calls(3N) 1096
svc_dg_enablecache(3N) 1096
svc_done(3N) 1096
svc_exit(3N) 1096
svc_fdset(3N) 1096
svc_freeargs(3N) 1096
svc_getargs(3N) 1096
svc_getreq_common(3N) 1096
svc_getreq_poll(3N) 1096
svc_getreqset(3N) 1096
svc_getrpccaller(3N) 1096
svc_max_pollfd(3N) 1096
svc_pollfd(3N) 1096
svc_run(3N) 1096
svc_sendreply(3N) 1096
rpc_svc_calls(3N) 1101
svc_dg_enablecache(3N) 1101
svc_done(3N) 1101
svc_exit(3N) 1101
svc_fdset(3N) 1101
svc_freeargs(3N) 1101
svc_getargs(3N) 1101
svc_getreq_common(3N) 1101

svc_getreq_poll(3N) 1101
svc_getreqset(3N) 1101
svc_getrpccaller(3N) 1101
svc_max_pollfd(3N) 1101
svc_pollfd(3N) 1101
svc_run(3N) 1101
svc_sendreply(3N) 1101
rpc_svc_create(3N) 1106
svc_control(3N) 1106
svc_create(3N) 1106
svc_destroy(3N) 1106
svc_dg_create(3N) 1106
svc_fd_create(3N) 1106
svc_raw_create(3N) 1106
svc_tli_create(3N) 1106
svc_tp_create(3N) 1106
svc_vc_create(3N) 1106
rpc_svc_reg(3N) 1111
rpc_reg(3N) 1111
svc_reg(3N) 1111
svc_unreg(3N) 1111
svc_auth_reg(3N) 1111
xpirt_register(3N) 1111
xpirt_unregister(3N) 1111
rpc_svc_calls(3N) 1114
svc_dg_enablecache(3N) 1114
svc_done(3N) 1114
svc_exit(3N) 1114

svc_fdset(3N) 1114
svc_freeargs(3N) 1114
svc_getargs(3N) 1114
svc_getreq_common(3N) 1114
svc_getreq_poll(3N) 1114
svc_getreqset(3N) 1114
svc_getrpccaller(3N) 1114
svc_max_pollfd(3N) 1114
svc_pollfd(3N) 1114
svc_run(3N) 1114
svc_sendreply(3N) 1114
rpc_svc_calls(3N) 1119
svc_dg_enablecache(3N) 1119
svc_done(3N) 1119
svc_exit(3N) 1119
svc_fdset(3N) 1119
svc_freeargs(3N) 1119
svc_getargs(3N) 1119
svc_getreq_common(3N) 1119
svc_getreq_poll(3N) 1119
svc_getreqset(3N) 1119
svc_getrpccaller(3N) 1119
svc_max_pollfd(3N) 1119
svc_pollfd(3N) 1119
svc_run(3N) 1119
svc_sendreply(3N) 1119
rpc_svc_create(3N) 1124
svc_control(3N) 1124

svc_create(3N) 1124
 svc_destroy(3N) 1124
 svc_dg_create(3N) 1124
 svc_fd_create(3N) 1124
 svc_raw_create(3N) 1124
 svc_tli_create(3N) 1124
 svc_tp_create(3N) 1124
 svc_vc_create(3N) 1124
 rpc_svc_create(3N) 1129
 svc_control(3N) 1129
 svc_create(3N) 1129
 svc_destroy(3N) 1129
 svc_dg_create(3N) 1129
 svc_fd_create(3N) 1129
 svc_raw_create(3N) 1129
 svc_tli_create(3N) 1129
 svc_tp_create(3N) 1129
 svc_vc_create(3N) 1129
 rpc_svc_reg(3N) 1134
 rpc_reg(3N) 1134
 svc_reg(3N) 1134
 svc_unreg(3N) 1134
 svc_auth_reg(3N) 1134
 xprt_register(3N) 1134
 xprt_unregister(3N) 1134
 rpc_svc_create(3N) 1137
 svc_control(3N) 1137
 svc_create(3N) 1137

svc_destroy(3N)	1137
svc_dg_create(3N)	1137
svc_fd_create(3N)	1137
svc_raw_create(3N)	1137
svc_tli_create(3N)	1137
svc_tp_create(3N)	1137
svc_vc_create(3N)	1137
t6alloc_blk(3N)	1142
t6free_blk(3N)	1142
t6attr_query(3N)	1143
t6clear_blk(3N)	1144
t6cmp_blk(3N)	1145
t6copy_blk(3N)	1146
t6dup_blk(3N)	1147
t6ext_attr(3N)	1148
t6new_attr(3N)	1148
t6alloc_blk(3N)	1149
t6free_blk(3N)	1149
t6get_attr(3N)	1150
t6set_attr(3N)	1150
t6get_endpt_mask(3N)	1152
t6set_endpt_mask(3N)	1152
t6get_endpt_default(3N)	1152
t6set_endpt_default(3N)	1152
t6get_endpt_mask(3N)	1154
t6set_endpt_mask(3N)	1154
t6get_endpt_default(3N)	1154
t6set_endpt_default(3N)	1154

t6peek_attr(3N) 1156
 t6last_attr(3N) 1156
 t6ext_attr(3N) 1157
 t6new_attr(3N) 1157
 t6peek_attr(3N) 1158
 t6last_attr(3N) 1158
 t6recvfrom(3N) 1159
 t6sendto(3N) 1161
 t6get_attr(3N) 1164
 t6set_attr(3N) 1164
 t6get_endpt_mask(3N) 1166
 t6set_endpt_mask(3N) 1166
 t6get_endpt_default(3N) 1166
 t6set_endpt_default(3N) 1166
 t6get_endpt_mask(3N) 1168
 t6set_endpt_mask(3N) 1168
 t6get_endpt_default(3N) 1168
 t6set_endpt_default(3N) 1168
 t6size_attr(3N) 1170
 t_accept(3N) 1171
 t_bind(3N) 1175
 t_optmgmt(3N) 1180
 t_snd(3N) 1187
 t_sndudata(3N) 1191
 labelbuilder(3) 1194
 tsol_lbuild_create(3) 1194
 tsol_lbuild_get(3) 1194
 tsol_lbuild_set(3) 1194

tsol_lbuild_destroy(3) 1194
labelbuilder(3) 1200
tsol_lbuild_create(3) 1200
tsol_lbuild_get(3) 1200
tsol_lbuild_set(3) 1200
tsol_lbuild_destroy(3) 1200
labelbuilder(3) 1206
tsol_lbuild_create(3) 1206
tsol_lbuild_get(3) 1206
tsol_lbuild_set(3) 1206
tsol_lbuild_destroy(3) 1206
labelbuilder(3) 1212
tsol_lbuild_create(3) 1212
tsol_lbuild_get(3) 1212
tsol_lbuild_set(3) 1212
tsol_lbuild_destroy(3) 1212
getutxent(3C) 1218
getutxid(3C) 1218
getutxline(3C) 1218
pututxline(3C) 1218
setutxent(3C) 1218
endutxent(3C) 1218
utmpxname(3C) 1218
getutmp(3C) 1218
getutmpx(3C) 1218
updwtmp(3C) 1218
updwtmpx(3C) 1218
getutxent(3C) 1222

getutxid(3C) 1222
getutxline(3C) 1222
pututxline(3C) 1222
setutxent(3C) 1222
endutxent(3C) 1222
utmpxname(3C) 1222
getutmp(3C) 1222
getutmpx(3C) 1222
updwtmp(3C) 1222
updwtmpx(3C) 1222
getutent(3C) 1226
getutid(3C) 1226
getutline(3C) 1226
pututline(3C) 1226
setutent(3C) 1226
endutent(3C) 1226
utmpname(3C) 1226
getutxent(3C) 1229
getutxid(3C) 1229
getutxline(3C) 1229
pututxline(3C) 1229
setutxent(3C) 1229
endutxent(3C) 1229
utmpxname(3C) 1229
getutmp(3C) 1229
getutmpx(3C) 1229
updwtmp(3C) 1229
updwtmpx(3C) 1229

labelclipping(3) 1233
 Xbcltos(3) 1233
 Xbsltos(3) 1233
 Xbiltos(3) 1233
 Xbcleartos(3) 1233
 labelclipping(3) 1235
 Xbcltos(3) 1235
 Xbsltos(3) 1235
 Xbiltos(3) 1235
 Xbcleartos(3) 1235
 labelclipping(3) 1237
 Xbcltos(3) 1237
 Xbsltos(3) 1237
 Xbiltos(3) 1237
 Xbcleartos(3) 1237
 labelclipping(3) 1239
 Xbcltos(3) 1239
 Xbsltos(3) 1239
 Xbiltos(3) 1239
 Xbcleartos(3) 1239
 rpc_svc_reg(3N) 1241
 rpc_reg(3N) 1241
 svc_reg(3N) 1241
 svc_unreg(3N) 1241
 svc_auth_reg(3N) 1241
 xpirt_register(3N) 1241
 xpirt_unregister(3N) 1241
 rpc_svc_reg(3N) 1244

rpc_reg(3N) 1244
 svc_reg(3N) 1244
 svc_unreg(3N) 1244
 svc_auth_reg(3N) 1244
 xprt_register(3N) 1244
 xprt_unregister(3N) 1244
 XTSOLgetClientAttributes(3X11TSOL) 1247
 XTSOLgetPropAttributes(3X11TSOL) 1248
 XTSOLgetPropLabel(3X11TSOL) 1249
 XTSOLgetPropUID(3X11TSOL) 1250
 XTSOLgetResAttributes(3X11TSOL) 1251
 XTSOLgetResLabel(3X11TSOL) 1252
 XTSOLgetResUID(3X11TSOL) 1253
 XTSOLgetSSHeight(3X11TSOL) 1254
 XTSOLgetWorkstationOwner(3X11TSOL) 1255
 XTSOLIsWindowTrusted(3X11TSOL) 1256
 XTSOLMakeTPWindow(3X11TSOL) 1257
 XTSOLsetPolyInstInfo(3X11TSOL) 1258
 XTSOLsetPropLabel(3X11TSOL) 1259
 XTSOLsetPropUID(3X11TSOL) 1261
 XTSOLsetResLabel(3X11TSOL) 1262
 XTSOLsetResUID(3X11TSOL) 1263
 XTSOLsetSessionHI(3X11TSOL) 1264
 XTSOLsetSessionLO(3X11TSOL) 1265
 XTSOLsetSSHeight(3X11TSOL) 1266
 XTSOLsetWorkstationOwner(3X11TSOL) 1267
 XTSOLShutdown(3X11TSOL) 1268
Index 1268

PREFACE

Overview

A man page is provided for both the naive user and the sophisticated user who is familiar with the Trusted Solaris operating environment and is in need of online information. 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.

Trusted Solaris Reference Manual

In the AnswerBook2™ and online man command forms of the man pages, all man pages are available:

- Trusted Solaris man pages that are unique for the Trusted Solaris environment
- SunOS 5.7 man pages that have been changed in the Trusted Solaris environment
- SunOS 5.7 man pages that remain unchanged.

The printed manual, the *Trusted Solaris 7 Reference Manual* contains:

- Man pages that have been added to the SunOS operating system by the Trusted Solaris environment
- Man pages that originated in SunOS 5.7, but have been modified in the Trusted Solaris environment to handle security requirements.

Users of printed manuals need both manuals in order to have a full set of man pages, since the *SunOS5.7 Reference Manual* contains the common man pages that are not modified in the Trusted Solaris environment.

Man Page Sections

The following contains a brief description of each section in the man pages 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 of this volume.
- 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 6 contains available games and demos.
- 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 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may 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 pathname 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:

- [] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
- . . . Ellipses. Several values may 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 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, functions and such, are described under USAGE.

IOCTL

This section appears on pages in Section 7 only. Only the device class which 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 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 is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below 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 root, `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 filenames 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.

SUMMARY OF TRUSTED SOLARIS CHANGES

This section describes changes to a Solaris 7 item by Trusted Solaris software. It is present in man pages that have been modified from Solaris 7 software.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications. The references are divided into two sections, so that users of printed manuals can easily locate a man page in its appropriate printed manual.

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.

C Library Functions

NAME	Intro – Introduction to functions and libraries
DESCRIPTION	<p>This section describes functions found in various libraries in the Trusted Solaris environment, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.</p> <p>These functions can be:</p> <ul style="list-style-type: none"> ■ Functions that are unique to and originate in the Trusted Solaris environment, such as <code>labelinfo(3)</code>. <code>labelinfo()</code> gets information about security labels from the <code>label_encodings(4)</code> file. ■ SunOS 5.7 functions and X windows functions that have been modified to work within the Trusted Solaris security policy, such as <code>accept(3N)</code>. Man pages for modified functions have been rewritten to remove information that is not accurate for how the function behaves in the Trusted Solaris environment. Modified man pages, such as <code>accept()</code>, also contain descriptions for any added features and arguments. ■ SunOS 5.7 functions that remain unchanged from the Solaris 7 release, such as <code>connect(3N)</code>. <hr/> <p>The printed <i>Trusted Solaris 7 Reference Manual</i> includes only those functions that have been modified or originate in the Trusted Solaris environment. This includes X Windows Library man pages, located in <code>/usr/openwin/man/man3x11tsol</code>. Printed versions of unchanged SunOS 5.7 man pages are found in the <i>SunOS 5.7 Reference Manual</i>.</p> <hr/> <p>Function declarations can be obtained from the <code>#include</code> files indicated on each page. Certain major collections are identified by a letter after the section number:</p> <p>(3B) These functions constitute the Source Compatibility (with BSD functions) library. It is implemented as a shared object, <code>libucb.so</code>, and as an archive, <code>libucb.a</code>, but is not automatically linked by the C compilation system. Specify <code>-lucb</code> on the <code>cc</code> command line to link with this library, which is located in the <code>/usr/ucb</code> subdirectory. Header files for this library are located within <code>/usr/ucbinclude</code>.</p> <p>(3C) These functions, together with those of Section 2 and those marked (3S), constitute the standard C library, <code>libc</code>, which is automatically linked by the C compilation system. The standard C library is implemented as a shared object, <code>libc.so</code>, and as an archive, <code>libc.a</code>. C programs are linked with the shared object version of the standard C library by default. Specify <code>-dn</code> on the <code>cc</code> command line to link with the archive version. See <code>libc(4)</code>, <code>cc(1B)</code> for other overrides, and the “C Compilation System” chapter of the <i>ANSI C Programmer’s Guide</i> for a discussion. Some functions behave differently in standard-conforming environments. This behavior is noted on the individual manual pages. See <code>standards(5)</code>.</p>

Some functions in `libc` have been modified for the Trusted Solaris environment. Changes in behavior or requirements are noted on the individual man pages.

- (3E) These functions constitute the ELF access library, `libelf`, (Extensible Linking Formats). This library provides the interface for the creation and analyses of “elf” files; executables, objects, and shared objects. `libelf` is implemented as a shared object, `libelf.so`, and as an archive, `libelf.a`, but is not automatically linked by the C compilation system. Specify `-lelf` on the `cc` command line to link with this library. See `libelf(4)`.
- (3G) These functions constitute the string pattern-matching & pathname manipulation library, `libgen`. This library is implemented as an archive, `libgen.a`, but not as a shared object, and is not automatically linked by the C compilation system. Specify `-lgen` on the `cc` command line to link with this library.
- (3K) These functions allow access to the kernel’s virtual memory library, which is implemented as a shared object, `libkvm.so`, and as an archive, `libkvm.a`, but is not automatically linked by the C compilation system. Specify `-lkvm` on the `cc` command line to link with this library. See `libkvm(4)`.

The `kstat_write()` function in `libkvm` has been modified for the Trusted Solaris environment. Changes in behavior or requirements are noted on the man page.
- (3M) These functions constitute the math library, `libm`. This library is implemented as a shared object, `libm.so`, and as an archive, `libm.a`, but is not automatically linked by the C compilation system. Specify `-lm` on the `cc` command line to link with this library. See `libm(4)`.
- (3N) These functions constitute the Network Service Library, `libnsl`. See `libnsl(4)`. The Trusted Solaris environment modifies some Network Service Library functions, and adds the Trusted Systems Interoperability Group (TSIG) TSIX [RE]1.1 library, `libt6` to the section. See `libt6(3)`.

`libnsl.so` and `libt6.so` are implemented as shared objects, and `libnsl.a` is also specified as an archive. Neither library is automatically linked by the C compilation system. Specify `-lnsl` on the `cc` command line to link with the `libnsl`. Specify `-lt6` on the `cc` command line to link with the `libt6` library.

Some of the functions documented in `man3n` incorporate other network libraries, including:
 - `libsocket` [see `libsocket(4)`],
 - `libresolv` [see `libresolv(4)`],
 - `librpcsvc` [see `librpcsvc(4)`],
 - `libnisdb` [see `libnisdb(4)`],

- `librac` [see `librac(4)`],
- `libxfn` [see `libxfn(4)`], and
- `libkrb` [see `libkrb(4)`].

Many base networking functions are also available in the X/Open Networking Interfaces library, `libxnet`. See section (3XN) below for more information on the `libxnet` interfaces.

Under all circumstances, the use of the Sockets API is recommended over the XTI and TLI APIs. If portability to other XPGV4v2 systems is a requirement, the application must use the `libxnet` interfaces. If portability is not required, the sockets interfaces in `libsocket` and `libnsl` are recommended over those in `libxnet`. Between the XTI and TLI APIs, the XTI interfaces (available with `libxnet`) are recommended over the TLI interfaces (available with `libnsl`).

- (3R) These functions constitute the POSIX.4 Realtime library, `librt`. It is implemented only as a shared object, `librt.so`, and is not automatically linked by the C compilation system. Specify `-lrt` on the `cc` command line to link with this library. Note that the former name for this library, `libposix4`, is maintained for backward compatibility but should be avoided. See `librt(4)`.

The `clock_gettime()` function in `librt` has been modified for the Trusted Solaris environment. Changes in behavior or requirements are noted on the man page.

- (3S) These functions constitute the “standard I/O package” [see `stdio(3S)`]. They can be compiled using the standard C library, `libc`, which is automatically linked by the C compilation system. The standard C library is implemented as a shared object, `libc.so`, and as an archive, `libc.a`. See `libc(4)`.

- (3T) These functions constitute the threads libraries, `libpthreads` and `libthread`. These libraries are used for building multithreaded applications. `libpthreads` implements the POSIX [see `standards(5)`] threads interface, whereas `libthread` implements the Solaris threads interface.

Both POSIX threads and Solaris threads can be used within the same application. Their implementations are completely compatible with each other; however, only POSIX threads guarantee portability to other POSIX-conforming environments.

When POSIX and Solaris threads are used in the same application, if there are calls with the same name but different semantics, the POSIX semantic supersedes the Solaris threads semantic. For example, the call to `fork()` will imply the `fork1()` semantic in a program linked with the POSIX threads library, whether or not it is also linked with `-lthread` (Solaris threads).

The `libpthreads` and `libthread` libraries are implemented as shared objects, `libpthreads.so` and `libthread.so`, respectively, but not as archived

libraries. `libpthread` and `libthread` are not automatically linked by the C compilation system. Specify `-lpthread` or `-lthread` on the `cc` command line to link with these libraries. See `libpthread(4)` and `libthread(4)`.

The following functions are optional under POSIX and are not supported in the current Solaris release.

```
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);
int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
*attr, int *protocol);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t
*attr, int prioceiling);
int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
*attr, int *prioceiling);
```

- (3) These functions constitute the Trusted Solaris library `libtsol`. `libtsol.so` is implemented as a shared object but is not automatically linked by the C compilation system. To link with the `libtsol` library specify `-ltsol` on the `cc` command line.

- (3X) Specialized libraries. These functions are contained in libraries including, but not limited to,

- `libadm` [see `libadm(4)`],
- `libbsdmalloc` [see `libbsdmalloc(4)`],
- `libcrypt` [see `libcrypt(4)`],
- `libcurses` [see `libcurses(4)`],
- `libdl` [see `libdl(4)`],
- `libform` [see `libform(4)`],
- `libmail`,
- `libmalloc` [see `libmalloc(4)`],
- `libmapmalloc` [see `libmapmalloc(4)`],
- `libmenu` [see `libmenu(4)`], and
- `libpanel` [see `libpanel(4)`].

- (3X11TSOL) These functions constitute the Trusted Solaris extension to the X windows library `libXtsol`. `libXtsol.so` is implemented as a shared object but is not automatically linked by the C compilation system. To link with the `libXtsol` library, specify `-lXtsol` after `-lX11` on the `cc` command line (`cc -lX11 -lXtsol`).

- (3XC) These functions constitute the X/Open Curses library, located in `/usr/xpg4/lib/libcurses.so.1`. This library provides a set of

internationalized functions and macros for creating and modifying input and output to a terminal screen. Included in this library are functions for creating windows, highlighting text, writing to the screen, reading from user input, and moving the cursor. X/Open Curses is designed to optimize screen update activities. The X/Open Curses library conforms fully with Issue 4 of the X/Open Extended Curses specification.

(3XN)

These functions constitute X/Open networking interfaces which comply with the X/Open CAE Specification, Networking Services, Issue 4 (September, 1994), and are located in `/usr/lib/libxnet.so.1`. See `libxnet(4)` and `standards(5)` for compilation information.

SECURITY POLICY

System calls enforce policy for library routines, and you should generally look to the system call man page for the to find out how policy is enforced for the system call. However, policy is sometimes explained on the library routine man pages, according to the following guidelines:

- If the relationship between the library routine and the underlying system call is intuitively obvious, as is the relationship between `fopen(3)` and `open(2)`, the related system call is mentioned in the `SEE ALSO` section, and the policy is not repeated on the library routine's man page.
- If the relationship between the library routine and the underlying system call(s) is not obvious, the policy information appears on the library routine's man page.
- If the system call man page has so much information that the developer may have trouble finding it, the relevant information is repeated on the library routine's man page. An example is `t6peek_attr(3N)`, which relies on `streamio(7I)`, whose man page is 21 pages.
- If the library is the exposed interface, and if the system call is undocumented, the policy appears on the library man page. One example of this is in the TSIX library routines, some of which rely on undocumented system calls.

DEFINITIONS

character

A character is any bit pattern able to fit into a byte on the machine. *Exception:* in some international languages, a "character" may require more than one byte, and is represented in multi-bytes.

null character

The null character is a character with value 0, conventionally represented in the C language as `\0`. A character array is a sequence of characters. A null-terminated character array (a *string*) is a sequence of characters, the last of which is the null character. The null string is a character array containing only the terminating null character. A null pointer is the value that is obtained by casting 0 into a pointer. C guarantees that this value will not match that of any

legitimate pointer, so many functions that return pointers return `NULL` to indicate an error. The macro `NULL` is defined in `<stdio.h>`. Types of the form `size_t` are defined in the appropriate headers.

MT-Level of Libraries

See `attributes(5)` for descriptions of library MT-Levels.

FILES

INCDIR usually `/usr/include`
LIBDIR usually `/usr/ccs/lib`
LIBDIR/libc.so
LIBDIR/libc.a
LIBDIR/libgen.a
LIBDIR/libm.a
LIBDIR/libsfm.sa
`/usr/lib/libc.so.1`

SEE ALSO

Trusted Solaris 7
Reference Manual

For assistance specific to Trusted Solaris libraries, see `intro(2)`, specifically the `DEFINITIONS` section, and the *Trusted Solaris Developer's Guide*.

`fork(2)`

SunOS 5.7 Reference Manual

`ar(1)`, `cc(1B)`, `ld(1)`, `nm(1)`, `stdio(3S)`, `pthread_atfork(3T)`, `libadm(4)`, `libbsdmalloc(4)`, `libc(4)`, `libcrypt(4)`, `libcurses(4)`, `libdl(4)`, `libelf(4)`, `libform(4)`, `libkvm(4)`, `libmalloc(4)`, `libmapmalloc(4)`, `libmenu(4)`, `libmp(4)`, `libnisdb(4)`, `libnsl(4)`, `libpanel(4)`, `librac(4)`, `libresolv(4)`, `librpcsvc(4)`, `libsocket(4)`, `libpthread(4)`, `libthread(4)`, `libxfn(4)`, `libxnet(4)`, `attributes(5)`, `standards(5)`

Linker and Libraries Guide

Profiling Tools

ANSI C Programmer's Guide

DIAGNOSTICS

For functions that return floating-point values, error handling varies according to compilation mode. Under the `-xt` (default) option to `cc`, these functions return the conventional values `0`, `±HUGE`, or `NaN` when the function is undefined for the given arguments or when the value is not representable. In the `-xa` and `-xc` compilation modes, `±HUGE_VAL` is returned in stead of `±HUGE`. (`HUGE_VAL` and `HUGE` are defined in `<math.h>` to be infinity and the largest-magnitude single-precision number, respectively.)

NOTES ON MULTITHREADED APPLICATIONS

When compiling a multithreaded application, either the `_POSIX_C_SOURCE`, `_POSIX_PTHREAD_SEMANTICS`, or `_REENTRANT` flag must be defined on the command line. This enables special definitions for functions only applicable

to multithreaded applications. For POSIX.1c-conforming applications, define the `_POSIX_C_SOURCE` flag to be `>= 199506L`:

```
cc [flags] file... -D_POSIX_C_SOURCE=199506L -lpthread
```

For POSIX behavior with the Solaris `fork()` and `fork1()` distinction, compile as follows:

```
cc [flags] file... -D_POSIX_PTHREAD_SEMANTICS -lpthread
```

For Solaris threads behavior, compile as follows:

```
cc [flags] file... -D_REENTRANT -lpthread
```

When building a singlethreaded application, the above flags should be undefined. This generates a binary that is executable on previous Solaris releases, which do not support multithreading.

Unsafe interfaces should be called only from the main thread to ensure the application's safety.

MT-Safe interfaces are denoted in the `ATTRIBUTES` section of the functions and libraries manual pages [see `attributes(5)`]. If a manual page does not state explicitly that an interface is MT-Safe, the user should assume that the interface is unsafe.

REALTIME APPLICATIONS

Be sure to have set the environment variable `LD_BIND_NOW` to a non-null value to enable early binding. Refer to the “When Relocations are Processed” chapter in *Linker and Libraries Guide* for additional information.

NOTES

None of the functions, external variables, or macros should be redefined in the user's programs. Any other name may be redefined without affecting the behavior of other library functions, but such redefinition may conflict with a declaration in an included header.

The headers in *INCDIR* provide function prototypes (function declarations including the types of arguments) for most of the functions listed in this manual. Function prototypes allow the compiler to check for correct usage of these functions in the user's program. The `lint` program checker may also be used and will report discrepancies even if the headers are not included with `#include` statements. Definitions for Sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the `-l` option to

`lint`. (For example, `-lm` includes definitions for `libm`.) Use of `lint` is highly recommended. See the `lint` chapter in *Performance Profiling Tools*.

Users should carefully note the difference between STREAMS and *stream*. STREAMS is a set of kernel mechanisms that support the development of network services and data communication drivers. It is composed of utility routines, kernel facilities, and a set of data structures. A *stream* is a file with its associated buffering. It is declared to be a pointer to a type `FILE` defined in `<stdio.h>`.

In detailed definitions of components, it is sometimes necessary to refer to symbolic names that are implementation-specific, but which are not necessarily expected to be accessible to an application program. Many of these symbolic names describe boundary conditions and system limits.

In this section, for readability, these implementation-specific values are given symbolic names. These names always appear enclosed in curly brackets to distinguish them from symbolic names of other implementation-specific constants that are accessible to application programs by headers. These names are not necessarily accessible to an application program through a header, although they may be defined in the documentation for a particular system.

In general, a portable application program should not refer to these symbolic names in its code. For example, an application program would not be expected to test the length of an argument list given to a routine to determine if it was greater than `ARG_MAX`.

Name	Description
<code>XTSOLIsWindowTrusted(3X11TSOL)</code>	Test if a window is created by a trusted client
<code>XTSOLMakeTPWindow(3X11TSOL)</code>	Make this window a Trusted Path window
<code>XTSOLShutdown(3X11TSOL)</code>	Shut down the system
<code>XTSOLgetClientAttributes(3X11TSOL)</code>	Get all CMW attributes associated with a client
<code>XTSOLgetPropAttributes(3X11TSOL)</code>	Get all CMW attributes associated with a property hanging on a window
<code>XTSOLgetPropLabel(3X11TSOL)</code>	Get the CMW label associated with a property hanging on a window

XTSOLgetPropUID(3X11TSOL)	Get the UID associated with a property hanging on a window
XTSOLgetResAttributes(3X11TSOL)	Get all CMW attributes associated with a window or a pixmap
XTSOLgetResLabel(3X11TSOL)	Get the CMW label associated with a window, a pixmap, or a colormap
XTSOLgetResUID(3X11TSOL)	Get the UID associated with a window, a pixmap
XTSOLgetSSHeight(3X11TSOL)	Get the height of screen stripe
XTSOLgetWorkstationOwner(3X11TSOL)	Get the ownership of the workstation
XTSOLsetPolyInstInfo(3X11TSOL)	Set polyinstantiation information
XTSOLsetPropLabel(3X11TSOL)	Set the CMW label associated with a property hanging on a window
XTSOLsetPropUID(3X11TSOL)	Set the UID associated with a property hanging on a window
XTSOLsetResLabel(3X11TSOL)	Set the CMW label associated with a window or a pixmap
XTSOLsetResUID(3X11TSOL)	Set the UID associated with a window, a pixmap, or a colormap
XTSOLsetSSHeight(3X11TSOL)	Set the height of screen stripe
XTSOLsetSessionHI(3X11TSOL)	Set the session high sensitivity label to the window server
XTSOLsetSessionLO(3X11TSOL)	Set the session low sensitivity label to the window server
XTSOLsetWorkstationOwner(3X11TSOL)	Set the ownership of the workstation
Xbcleartos(3)	See labelclipping(3)
Xbcltos(3)	See labelclipping(3)
Xbiltos(3)	See labelclipping(3)

xbsltos(3)	See labelclipping(3)
accept(3N)	Accept a connection on a socket
adornfc(3)	Adorn the final component of a pathname
au_preselect(3)	Preselect an audit event
au_user_mask(3)	Get user's binary preselection mask
auditwrite(3)	Construct and write user-level audit records
auth_set_to_str(3)	See auth_to_str(3)
auth_to_str(3)	Translate and verify user authorizations
aw_errno(3)	See aw_strerror(3)
aw_perror(3)	See aw_strerror(3)
aw_strerror(3)	Obtain and display error messages
bclearhigh(3)	See blmanifest(3)
bclearlow(3)	See blmanifest(3)
bcleartoh(3)	See btohex(3)
bcleartoh_r(3)	See btohex(3)
bcleartos(3)	See bltos(3)
bclearundef(3)	See blmanifest(3)
bclearvalid(3)	See blvalid(3)
bclhigh(3)	See blmanifest(3)
bcllow(3)	See blmanifest(3)
bcltobanner(3)	Translate binary CMW labels to character-coded labels for a printer banner page
bcltoh(3)	See btohex(3)
bcltoh_r(3)	See btohex(3)
bcltoil(3)	See blportion(3)

bcltos(3)	See bltos(3)
bcltosl(3)	See blportion(3)
bclundef(3)	See blmanifest(3)
bilconjoin(3)	Conjoin binary information labels
bildominates(3)	See blcompare(3)
bilequal(3)	See blcompare(3)
bilhigh(3)	See blmanifest(3)
billow(3)	See blmanifest(3)
biltoh(3)	See btohex(3)
biltoh_r(3)	See btohex(3)
biltolev(3)	See blportion(3)
biltos(3)	See bltos(3)
bilundef(3)	See blmanifest(3)
bilvalid(3)	See blvalid(3)
bimdominates(3)	See blcompare(3)
bimequal(3)	See blcompare(3)
bind(3N)	Bind a name to a socket
blcompare(3)	Compare binary labels
bldominates(3)	See blcompare(3)
blequal(3)	See blcompare(3)
blinrange(3)	See blcompare(3)
blinset(3)	Check binary label for inclusion in set
blmanifest(3)	Create manifest binary labels
blmaximum(3)	See blminmax(3)
blminimum(3)	See blminmax(3)
blminmax(3)	Bound of two binary levels
blportion(3)	Access binary label portions

<code>blstrictdom(3)</code>	See <code>blcompare(3)</code>
<code>bltocolor(3)</code>	Get character-coded color name of label
<code>bltocolor_r(3)</code>	See <code>bltocolor(3)</code>
<code>bltos(3)</code>	Translate binary labels to character coded labels
<code>bltype(3)</code>	Compare and set the type of binary label
<code>blvalid(3)</code>	Check validity of binary label
<code>bslhigh(3)</code>	See <code>blmanifest(3)</code>
<code>bsllow(3)</code>	See <code>blmanifest(3)</code>
<code>bsltoh(3)</code>	See <code>btohex(3)</code>
<code>bsltoh_r(3)</code>	See <code>btohex(3)</code>
<code>bsltos(3)</code>	See <code>bltos(3)</code>
<code>bslundef(3)</code>	See <code>blmanifest(3)</code>
<code>bslvalid(3)</code>	See <code>blvalid(3)</code>
<code>btohex(3)</code>	Convert binary label to hexadecimal
<code>chkauth(3)</code>	See <code>auth_to_str(3)</code>
<code>clnt_call(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>clnt_control(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_create(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_create_timed(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_create_vers(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_create_vers_timed(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_destroy(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_dg_create(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_freeres(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>clnt_geterr(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>clnt_pcreateerror(3N)</code>	See <code>rpc_clnt_create(3N)</code>

<code>clnt_perrno(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>clnt_perror(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>clnt_raw_create(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_spcreateerror(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_sperrno(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>clnt_sperror(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>clnt_tli_create(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_tp_create(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_tp_create_timed(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clnt_vc_create(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>clock_getres(3R)</code>	See <code>clock_settime(3R)</code>
<code>clock_gettime(3R)</code>	See <code>clock_settime(3R)</code>
<code>clock_settime(3R)</code>	High-resolution clock operations
<code>dn_comp(3N)</code>	See <code>resolver(3N)</code>
<code>dn_expand(3N)</code>	See <code>resolver(3N)</code>
<code>door_create(3X)</code>	Create a door descriptor
<code>door_tcred(3X)</code>	Return the extended credential information associated with the client of the current door invocation
<code>endac(3)</code>	See <code>getacinfo(3)</code>
<code>endauclass(3)</code>	See <code>getauclassent(3)</code>
<code>endauevent(3)</code>	See <code>getauevent(3)</code>
<code>endauuser(3)</code>	See <code>getauusernam(3)</code>
<code>endprofent(3)</code>	See <code>getprofent(3)</code>
<code>endprofstr(3)</code>	See <code>getprofstr(3)</code>
<code>enduserent(3)</code>	See <code>getuserent(3)</code>
<code>endutent(3C)</code>	See <code>getutent(3C)</code>
<code>endutxent(3C)</code>	See <code>getutxent(3C)</code>

<code>free_auth_set(3)</code>	See <code>auth_to_str(3)</code>
<code>free_profent(3)</code>	See <code>getprofent(3)</code>
<code>free_profstr(3)</code>	See <code>getprofstr(3)</code>
<code>free_userent(3)</code>	See <code>getuserent(3)</code>
<code>ftw(3C)</code>	Walk a file tree
<code>get_auth_text(3)</code>	See <code>auth_to_str(3)</code>
<code>get_priv_text(3)</code>	See <code>priv_to_str(3)</code>
<code>getacdir(3)</code>	See <code>getacinfo(3)</code>
<code>getacflg(3)</code>	See <code>getacinfo(3)</code>
<code>getacinfo(3)</code>	Get audit control file information
<code>getacmin(3)</code>	See <code>getacinfo(3)</code>
<code>getacna(3)</code>	See <code>getacinfo(3)</code>
<code>getauclassent(3)</code>	get audit_class entry
<code>getauclassent_r(3)</code>	See <code>getauclassent(3)</code>
<code>getauclassnam(3)</code>	See <code>getauclassent(3)</code>
<code>getauclassnam_r(3)</code>	See <code>getauclassent(3)</code>
<code>getauditflags(3)</code>	Convert audit flag specifications
<code>getauditflagsbin(3)</code>	See <code>getauditflags(3)</code>
<code>getauditflagschar(3)</code>	See <code>getauditflags(3)</code>
<code>getauevent(3)</code>	Get audit_event entry
<code>getauevent_r(3)</code>	See <code>getauevent(3)</code>
<code>getauevnam(3)</code>	See <code>getauevent(3)</code>
<code>getauevnam_r(3)</code>	See <code>getauevent(3)</code>
<code>getauevnonam(3)</code>	See <code>getauevent(3)</code>
<code>getauevnum(3)</code>	See <code>getauevent(3)</code>
<code>getauevnum_r(3)</code>	See <code>getauevent(3)</code>
<code>getauuserent(3)</code>	See <code>getauusernam(3)</code>
<code>getauusernam(3)</code>	Get audit_user entry

getcil(3)	See blportion(3)
getcsl(3)	See blportion(3)
getfauditflags(3)	Generates the process audit state
getpeerinfo(3)	Get peer's process characteristics.
getprofent(3)	Get Trusted Solaris user profile description
getprofentbyname(3)	See getprofent(3)
getprofstr(3)	Get Trusted Solaris user profile description
getprofstrbyname(3)	See getprofstr(3)
getsockopt(3N)	Get and set options on sockets
getuserent(3)	Get Trusted Solaris user security attributes
getuserentbyname(3)	See getuserent(3)
getuserentbyuid(3)	See getuserent(3)
getutent(3C)	Access utmp file entry
getutid(3C)	See getutent(3C)
getutline(3C)	See getutent(3C)
getutmp(3C)	See getutxent(3C)
getutmpx(3C)	See getutxent(3C)
getutxent(3C)	Access utmpx file entry
getutxid(3C)	See getutxent(3C)
getutxline(3C)	See getutxent(3C)
getvfsaent(3)	Get vfstab_adjunct file entry
getvfsafile(3)	See getvfsaent(3)
h_alloc(3)	See btohex(3)
h_free(3)	See btohex(3)
hextob(3)	Convert hexadecimal string to binary label

htobcl(3)	See hextob(3)
htobclear(3)	See hextob(3)
htobil(3)	See hextob(3)
htobsl(3)	See hextob(3)
initgroups(3C)	Initialize the supplementary group access list
kstat_read(3K)	Read or write kstat data
kstat_write(3K)	See kstat_read(3K)
labelbuilder(3)	Create a Motif-based user interface for interactively building a valid label or clearance
labelclipping(3)	Translate a binary label and clip to the specified width
labelinfo(3)	Get information about the label encodings
labelvers(3)	Get version of the label_encodings file
libt6(3N)	TSIX trusted IPC library
listen(3N)	Listen for connections on a socket
mldgetcwd(3)	Get pathname of current working directory
mldlstat(3)	See mldstat(3)
mldrealpath(3)	Return the canonicalized absolute pathname, including any MLD adornments and SLD names
mldrealpath1(3)	See mldrealpath(3)
mldstat(3)	Get file status in multilevel directory
mlock(3C)	Lock or unlock pages in memory
mlockall(3C)	Lock or unlock address space

<code>munlock(3C)</code>	See <code>mlock(3C)</code>
<code>munlockall(3C)</code>	See <code>mlockall(3C)</code>
<code>nftw(3C)</code>	See <code>ftw(3C)</code>
<code>nis_add(3N)</code>	See <code>nis_names(3N)</code>
<code>nis_add_entry(3N)</code>	See <code>nis_tables(3N)</code>
<code>nis_addmember(3N)</code>	See <code>nis_groups(3N)</code>
<code>nis_checkpoint(3N)</code>	See <code>nis_ping(3N)</code>
<code>nis_creategroup(3N)</code>	See <code>nis_groups(3N)</code>
<code>nis_destroygroup(3N)</code>	See <code>nis_groups(3N)</code>
<code>nis_first_entry(3N)</code>	See <code>nis_tables(3N)</code>
<code>nis_freeresult(3N)</code>	See <code>nis_names(3N)</code>
<code>nis_freeservlist(3N)</code>	See <code>nis_server(3N)</code>
<code>nis_freetags(3N)</code>	See <code>nis_server(3N)</code>
<code>nis_getservlist(3N)</code>	See <code>nis_server(3N)</code>
<code>nis_groups(3N)</code>	NIS+ group manipulation functions
<code>nis_ismember(3N)</code>	See <code>nis_groups(3N)</code>
<code>nis_list(3N)</code>	See <code>nis_tables(3N)</code>
<code>nis_lookup(3N)</code>	See <code>nis_names(3N)</code>
<code>nis_mkdir(3N)</code>	See <code>nis_server(3N)</code>
<code>nis_modify(3N)</code>	See <code>nis_names(3N)</code>
<code>nis_modify_entry(3N)</code>	See <code>nis_tables(3N)</code>
<code>nis_names(3N)</code>	NIS+ namespace functions
<code>nis_next_entry(3N)</code>	See <code>nis_tables(3N)</code>
<code>nis_ping(3N)</code>	Misc NIS+ log administration functions
<code>nis_print_group_entry(3N)</code>	See <code>nis_groups(3N)</code>
<code>nis_remove(3N)</code>	See <code>nis_names(3N)</code>
<code>nis_remove_entry(3N)</code>	See <code>nis_tables(3N)</code>

<code>nis_removemember(3N)</code>	See <code>nis_groups(3N)</code>
<code>nis_rmdir(3N)</code>	See <code>nis_server(3N)</code>
<code>nis_server(3N)</code>	Miscellaneous NIS+ functions
<code>nis_servstate(3N)</code>	See <code>nis_server(3N)</code>
<code>nis_stats(3N)</code>	See <code>nis_server(3N)</code>
<code>nis_tables(3N)</code>	NIS+ table functions
<code>nis_verifygroup(3N)</code>	See <code>nis_groups(3N)</code>
<code>plock(3)</code>	Lock or unlock into memory process, text, or data
<code>priv_set_to_str(3)</code>	See <code>priv_to_str(3)</code>
<code>priv_to_str(3)</code>	Convert a numeric privilege to its name or a privilege name to its number
<code>putprofstr(3)</code>	See <code>getprofstr(3)</code>
<code>pututline(3C)</code>	See <code>getutent(3C)</code>
<code>pututxline(3C)</code>	See <code>getutxent(3C)</code>
<code>randomword(3)</code>	Generate random pronounceable password
<code>res_init(3N)</code>	See <code>resolver(3N)</code>
<code>res_mkquery(3N)</code>	See <code>resolver(3N)</code>
<code>res_mkupdate(3N)</code>	See <code>resolver(3N)</code>
<code>res_mkupdrec(3N)</code>	See <code>resolver(3N)</code>
<code>res_query(3N)</code>	See <code>resolver(3N)</code>
<code>res_search(3N)</code>	See <code>resolver(3N)</code>
<code>res_send(3N)</code>	See <code>resolver(3N)</code>
<code>res_update(3N)</code>	See <code>resolver(3N)</code>
<code>resolver(3N)</code>	Resolver routines
<code>rpc(3N)</code>	Library routines for remote procedure calls
<code>rpc_broadcast(3N)</code>	See <code>rpc_clnt_calls(3N)</code>

<code>rpc_broadcast_exp(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>rpc_call(3N)</code>	See <code>rpc_clnt_calls(3N)</code>
<code>rpc_clnt_calls(3N)</code>	Library routines for client side calls
<code>rpc_clnt_create(3N)</code>	Library routines for dealing with creation and manipulation of CLIENT handles
<code>rpc_createerr(3N)</code>	See <code>rpc_clnt_create(3N)</code>
<code>rpc_reg(3N)</code>	See <code>rpc_svc_reg(3N)</code>
<code>rpc_svc_calls(3N)</code>	Library routines for RPC servers
<code>rpc_svc_create(3N)</code>	Library routines for the creation of server handles
<code>rpc_svc_reg(3N)</code>	Library routines for registering servers
<code>rpcb_getaddr(3N)</code>	See <code>rpcbind(3N)</code>
<code>rpcb_getallmaps(3N)</code>	See <code>rpcbind(3N)</code>
<code>rpcb_getmaps(3N)</code>	See <code>rpcbind(3N)</code>
<code>rpcb_gettime(3N)</code>	See <code>rpcbind(3N)</code>
<code>rpcb_rmtcall(3N)</code>	See <code>rpcbind(3N)</code>
<code>rpcb_set(3N)</code>	See <code>rpcbind(3N)</code>
<code>rpcb_unset(3N)</code>	See <code>rpcbind(3N)</code>
<code>rpcbind(3N)</code>	Library routines for RPC bind service
<code>sbcleartos(3)</code>	See <code>sbltos(3)</code>
<code>sbcltos(3)</code>	See <code>sbltos(3)</code>
<code>sbiltos(3)</code>	See <code>sbltos(3)</code>
<code>sbltos(3)</code>	Translate binary labels to canonical character-coded labels
<code>sbsltos(3)</code>	See <code>sbltos(3)</code>
<code>send(3N)</code>	Send a message from a socket
<code>sendmsg(3N)</code>	See <code>send(3N)</code>

<code>sendto(3N)</code>	See <code>send(3N)</code>
<code>set_effective_priv(3)</code>	Assign a privilege set for the current process
<code>set_inheritable_priv(3)</code>	See <code>set_effective_priv(3)</code>
<code>set_permitted_priv(3)</code>	See <code>set_effective_priv(3)</code>
<code>setac(3)</code>	See <code>getacinfo(3)</code>
<code>setauclass(3)</code>	See <code>getauclassent(3)</code>
<code>setauevent(3)</code>	See <code>getauevent(3)</code>
<code>setauuser(3)</code>	See <code>getauusernam(3)</code>
<code>setbltype(3)</code>	See <code>bltype(3)</code>
<code>setcil(3)</code>	See <code>blportion(3)</code>
<code>setcsl(3)</code>	See <code>blportion(3)</code>
<code>setprofent(3)</code>	See <code>getprofent(3)</code>
<code>setprofstr(3)</code>	See <code>getprofstr(3)</code>
<code>setsockopt(3N)</code>	See <code>getsockopt(3N)</code>
<code>setuserent(3)</code>	See <code>getuserent(3)</code>
<code>setutent(3C)</code>	See <code>getutent(3C)</code>
<code>setutxent(3C)</code>	See <code>getutxent(3C)</code>
<code>stobcl(3)</code>	See <code>stobl(3)</code>
<code>stobclear(3)</code>	See <code>stobl(3)</code>
<code>stobil(3)</code>	See <code>stobl(3)</code>
<code>stobl(3)</code>	Translate character-coded labels to binary labels
<code>stobsl(3)</code>	See <code>stobl(3)</code>
<code>str_to_auth(3)</code>	See <code>auth_to_str(3)</code>
<code>str_to_auth_set(3)</code>	See <code>auth_to_str(3)</code>
<code>str_to_priv(3)</code>	See <code>priv_to_str(3)</code>
<code>str_to_priv_set(3)</code>	See <code>priv_to_str(3)</code>
<code>svc_auth_reg(3N)</code>	See <code>rpc_svc_reg(3N)</code>

<code>svc_control(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_create(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_destroy(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_dg_create(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_dg_enablecache(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_done(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_exit(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_fd_create(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_fdset(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_freeargs(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_getargs(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_getreq_common(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_getreq_poll(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_getreqset(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_getrpccaller(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_max_pollfd(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_pollfd(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_raw_create(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_reg(3N)</code>	See <code>rpc_svc_reg(3N)</code>
<code>svc_run(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_sendreply(3N)</code>	See <code>rpc_svc_calls(3N)</code>
<code>svc_tli_create(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_tp_create(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>svc_unreg(3N)</code>	See <code>rpc_svc_reg(3N)</code>
<code>svc_vc_create(3N)</code>	See <code>rpc_svc_create(3N)</code>
<code>t6alloc_blk(3N)</code>	Allocate and free security-attribute control structure and buffer
<code>t6attr_query(3N)</code>	Get mask indicating which attributes came from templates

<code>t6clear_blk(3N)</code>	Clear security attributes
<code>t6cmp_blk(3N)</code>	Compare security attributes
<code>t6copy_blk(3N)</code>	Copy security attributes
<code>t6dup_blk(3N)</code>	Duplicate security attributes
<code>t6ext_attr(3N)</code>	Manipulate network-endpoint security options
<code>t6free_blk(3N)</code>	See <code>t6alloc_blk(3N)</code>
<code>t6get_attr(3N)</code>	Get security attributes from or set security attributes in the security-attribute buffer handled by a control structure
<code>t6get_endpt_default(3N)</code>	See <code>t6get_endpt_mask(3N)</code>
<code>t6get_endpt_mask(3N)</code>	Get and set endpoint mask, or get and set endpoint default attributes
<code>t6last_attr(3N)</code>	See <code>t6peek_attr(3N)</code>
<code>t6new_attr(3N)</code>	See <code>t6ext_attr(3N)</code>
<code>t6peek_attr(3N)</code>	Examine the security attributes on the next or the previous byte of data
<code>t6recvfrom(3N)</code>	Read security attributes and data from a trusted endpoint
<code>t6sendto(3N)</code>	Specify security attributes to send with data on a trusted endpoint
<code>t6set_attr(3N)</code>	See <code>t6get_attr(3N)</code>
<code>t6set_endpt_default(3N)</code>	See <code>t6get_endpt_mask(3N)</code>
<code>t6set_endpt_mask(3N)</code>	See <code>t6get_endpt_mask(3N)</code>
<code>t6size_attr(3N)</code>	Get the size of a particular attribute from the control structure
<code>t_accept(3N)</code>	Accept a connection request
<code>t_bind(3N)</code>	Bind an address to a transport endpoint

<code>t_optmgmt(3N)</code>	Manage options for a transport endpoint
<code>t_snd(3N)</code>	Send data or expedited data over a connection
<code>t_sndudata(3N)</code>	Send a data unit
<code>tsol_lbuild_create(3)</code>	See <code>labelbuilder(3)</code>
<code>tsol_lbuild_destroy(3)</code>	See <code>labelbuilder(3)</code>
<code>tsol_lbuild_get(3)</code>	See <code>labelbuilder(3)</code>
<code>tsol_lbuild_set(3)</code>	See <code>labelbuilder(3)</code>
<code>updwtmp(3C)</code>	See <code>getutxent(3C)</code>
<code>updwtmpx(3C)</code>	See <code>getutxent(3C)</code>
<code>utmpname(3C)</code>	See <code>getutent(3C)</code>
<code>utmpxname(3C)</code>	See <code>getutxent(3C)</code>
<code>xprt_register(3N)</code>	See <code>rpc_svc_reg(3N)</code>
<code>xprt_unregister(3N)</code>	See <code>rpc_svc_reg(3N)</code>

NAME	accept – Accept a connection on a socket						
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <sys/socket.h> int accept(int s, struct sockaddr *addr, socklen_t *addrlen);</pre>						
DESCRIPTION	<p>The argument <i>s</i> is a socket that has been created with <code>socket(3N)</code> and bound to an address with <code>bind(3N)</code>, and that is listening for connections after a call to <code>listen(3N)</code>. The <code>accept()</code> function extracts the first connection on the queue of pending connections, creates a new socket with the properties of <i>s</i>, and allocates a new file descriptor, <i>ns</i>, for the socket. If no pending connections are present on the queue and the socket is not marked as non-blocking, <code>accept()</code> blocks the caller until a connection is present. If the socket is marked as non-blocking and no pending connections are present on the queue, <code>accept()</code> returns an error as described below. The <code>accept()</code> function uses the <code>netconfig(4)</code> file to determine the STREAMS device file name associated with <i>s</i>. This is the device on which the connect indication will be accepted. The accepted socket, <i>ns</i>, is used to read and write data to and from the socket that connected to <i>ns</i>; it is not used to accept more connections. The original socket (<i>s</i>) remains open for accepting further connections.</p> <p>The argument <i>addr</i> is a result parameter that is filled in with the address of the connecting entity as it is known to the communications layer. The exact format of the <i>addr</i> parameter is determined by the domain in which the communication occurs.</p> <p>The argument <i>addrlen</i> is a value-result parameter. Initially, it contains the amount of space pointed to by <i>addr</i>; on return it contains the length in bytes of the address returned.</p> <p>The <code>accept()</code> function is used with connection-based socket types, currently with <code>SOCK_STREAM</code>.</p> <p>It is possible to <code>select(3C)</code> or <code>poll(2)</code> a socket for the purpose of an <code>accept()</code> by selecting or polling it for a read. However, this will only indicate when a connect indication is pending; it is still necessary to call <code>accept()</code>.</p>						
RETURN VALUES	The <code>accept()</code> function returns <code>-1</code> on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.						
ERRORS	<p><code>accept()</code> will fail if:</p> <table> <tr> <td>EBADF</td><td>The descriptor is invalid.</td></tr> <tr> <td>EINTR</td><td>The accept attempt was interrupted by the delivery of a signal.</td></tr> <tr> <td>EMFILE</td><td>The per-process descriptor table is full.</td></tr> </table>	EBADF	The descriptor is invalid.	EINTR	The accept attempt was interrupted by the delivery of a signal.	EMFILE	The per-process descriptor table is full.
EBADF	The descriptor is invalid.						
EINTR	The accept attempt was interrupted by the delivery of a signal.						
EMFILE	The per-process descriptor table is full.						

ENODEV	The protocol family and type corresponding to <code>s</code> could not be found in the <code>netconfig</code> file.
ENOMEM	There was insufficient user memory available to complete the operation.
ENOSR	There were insufficient STREAMS resources available to complete the operation.
ENOTSOCK	The descriptor does not reference a socket.
EOPNOTSUPP	The referenced socket is not of type <code>SOCK_STREAM</code> .
EPROTO	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized or the connection has already been released.
EWouldBlock	The socket is marked as non-blocking and no connections are present to be accepted.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the calling process possesses the `PRIV_NET_MAC_READ` privilege and the socket has been bound to a multilevel port (MLP), the connection is accepted on a MLP; otherwise, the connection is accepted on a single-level port (SLP). See `bind(3N)` for more information.

SEE ALSO

Trusted Solaris 7
Reference Manual

`bind(3N)`, `listen(3N)`

SunOS 5.7 Reference
Manual

`poll(2)`, `select(3C)`, `connect(3N)`, `socket(3N)`, `netconfig(4)`,
`attributes(5)`, `socket(5)`

NAME	adornfc – Adorn the final component of a pathname												
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/mld.h> int adornfc(char *path_name, char *adorned_name);</pre>												
DESCRIPTION	<p>adornfc() adorns the final component of <i>path_name</i> unless it is already adorned. <i>path_name</i> is a pathname to a filesystem object. <i>adorned_name</i> is a pointer to a buffer in which the adorned version of <i>path_name</i> is placed. This buffer should be of at least MAXPATHLEN bytes in length.</p>												
RETURN VALUES	<p>adornfc() returns:</p> <p>0 On success.</p> <p>-1 On failure and sets <code>errno</code> to indicate the error.</p>												
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe						
ATTRIBUTE TYPE	ATTRIBUTE VALUE												
Availability	SUNWtsu												
MT-Level	MT-Safe												
ERRORS	<p>adornfc() fails if one or more of the following are true:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path_name</i>.</td></tr> <tr> <td>EFAULT</td><td><i>path_name</i> or <i>adorned_name</i> points to an invalid address.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path_name</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the path argument exceeds <code>PATH_MAX</code> or <code>MAXPATHLEN</code>. A pathname component is longer than <code>NAME_MAX</code> (see <code>sysconf(3C)</code>) while <code>_POSIX_NO_TRUNC</code> is in effect (see <code>pathconf(2)</code>).</td></tr> <tr> <td>ENOENT</td><td>A component of the path prefix of <i>path_name</i> does not exist.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path_name</i> .	EFAULT	<i>path_name</i> or <i>adorned_name</i> points to an invalid address.	EIO	An I/O error occurred while reading from the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .	ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> or <code>MAXPATHLEN</code> . A pathname component is longer than <code>NAME_MAX</code> (see <code>sysconf(3C)</code>) while <code>_POSIX_NO_TRUNC</code> is in effect (see <code>pathconf(2)</code>).	ENOENT	A component of the path prefix of <i>path_name</i> does not exist.
EACCES	Search permission is denied for a component of the path prefix of <i>path_name</i> .												
EFAULT	<i>path_name</i> or <i>adorned_name</i> points to an invalid address.												
EIO	An I/O error occurred while reading from the file system.												
ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .												
ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> or <code>MAXPATHLEN</code> . A pathname component is longer than <code>NAME_MAX</code> (see <code>sysconf(3C)</code>) while <code>_POSIX_NO_TRUNC</code> is in effect (see <code>pathconf(2)</code>).												
ENOENT	A component of the path prefix of <i>path_name</i> does not exist.												

ENOTDIR

A component of the path prefix of *path_name* is not a directory.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

pathconf(2)

**SunOS 5.7 Reference
Manual**

attributes(5)

NAME	auditwrite – Construct and write user-level audit records
SYNOPSIS	<pre>cc [flag...] file... -lbsm -lsocket -lnsl -lintl -ltsol [library...] #include <bsm/auditwrite.h> #include <bsm/audit_uevents.h> #include <tsol/priv.h> #include <tsol/label.h> int auditwrite(...,AW_END);</pre>
DESCRIPTION	auditwrite() provides a single-function programmer interface to auditing for user-level programs. The principal features of auditwrite() are audit record construction, audit record queueing, a save area, and support for trusted server auditing. See NOTES for privileges needed to write audit records.
Record Construction	auditwrite() creates complete audit records or appends information to an existing, partial audit record. A single-shot audit record is one that is constructed and written in a single call to auditwrite(). Multi-shot audit records are those constructed piecemeal through two or more calls to auditwrite(). See EXAMPLES.
Record Queuing	Audit records may be queued by specifying a threshold. When the threshold is reached, records are written in one operation. This batching minimizes system-call overhead.
Save Area	A special audit-record buffer may be requested as a save area. Attributes stored in the save area are prepended to every subsequent record written with auditwrite().
Trusted Server Auditing Support	Some trusted servers act on behalf of untrusted client processes performing security checks and providing access to TCB objects. A trusted server must sometimes generate audit records with the audit characteristics of its clients. The AW_SERVER command tells auditwrite() that the caller is a trusted server and that additional information should be added to each audit record if the information has not already been provided.
PARAMETERS	auditwrite() takes a variable number of arguments. Arguments are of three types. One type, referred to as control commands, controls the behavior of auditwrite(). One and only one control command may appear on the auditwrite() invocation line. Another type of argument is an attribute command. Attribute commands describe the attributes that compose an audit record. Control and attribute commands may appear in any order in an auditwrite() invocation. The last type of argument is the terminator

Control Commands

command. The terminator command `AW_END` must be the last argument on the `auditwrite` invocation line. The terminator command notifies `auditwrite` when to stop parsing the invocation line.

`AW_WRITE` Write to the audit trail the default audit record or the audit record specified by the last `AW_USERD` command. If queueing is in effect, the record is queued and written when the queue is flushed.

`AW_APPEND` Attach one or more record attributes to the end of the record. `AW_APPEND` is used in the multi-shot construction of an audit record. Attributes are kept in a record buffer until `auditwrite()` is called with the `AW_WRITE` command. One or more attribute commands must be specified with the `AW_APPEND` command.

`AW_DEFAULTRD` Use the default record descriptor.

`AW_DISCARD` Discard all partial and complete audit records.

`AW_DISCARDRD, int rd` Discard the record descriptor specified by *rd*. An *rd* of `-1` can be specified to discard the default *rd*.

`AW_GETRD, int *rd` Obtain an audit record descriptor for use with `AW_USERD`.

`AW_PRESELECT, au_mask_t *pmask` Preselect audit records according to *pmask*. Normally audit records are preselected by `auditwrite()` using the preselection mask in the execution environment. See `getaudit(2)` for details.

`AW_NOPRESELECT` Use the preselection mask in the execution environment instead of the one specified with the `AW_PRESELECT` command.

`AW_QUEUE, u_int hi_water` Turn on audit-record queueing. `AW_QUEUE` causes `auditwrite()` to queue all audit records. When the total number of bytes of all queued audit records reaches *hi_water*, the queue is flushed. The queue may also be flushed at will with `AW_FLUSH`. The `auditwrite()` audit-record queue should not be confused with the kernel audit-record queue. The `auditwrite()` queue is a separate and distinct queue created in user

		address space. The <code>auditwrite()</code> audit-record queue can minimize system-call overhead by writing several audit records in one operation.
	<code>AW_NOQUEUE</code>	Turn off audit-record queueing. Flush the queue.
	<code>AW_FLUSH</code>	Flush the audit-record queue.
	<code>AW_SAVERD, int *rd</code>	Turn on use of a save area. Attributes stored in the save area are prepended to records before they are written.
	<code>AW_NOSAVE</code>	Turn off use of the save area.
	<code>AW_SERVER</code>	Turn on the trusted server option. The <code>AW_SERVER</code> command tells <code>auditwrite()</code> that the calling program is a server that must generate complete audit records. <code>auditwrite()</code> adds header and trailer attributes to all records. Sequence and group attributes are also added depending upon the audit policy. See <code>audit(2)</code> for further information on audit policy.
	<code>AW_NOSERVER</code>	Turn off the trusted server option.
	<code>AW_USERD, int rd</code>	Use the record descriptor obtained with <code>AW_GETRD</code> .
Attribute Commands	Attribute commands describe the attributes that compose an audit record. Attribute commands must be specified with one and only one <code>AW_APPEND</code> or <code>AW_WRITE</code> control command.	
	<code>AW_ARG, char n,</code> <code>char *text</code> <code>u_long v</code>	Place the specified system call argument information into the audit record. <i>n</i> contains the argument number. <i>text</i> contains a string describing the argument. <i>v</i> contains the value of the argument.
	<code>AW_ATTR, mode_t</code> <i>mode</i> , <code>uid_t uid</code> , <code>gid_t gid</code> , <code>dev_t dev</code> , <code>ino_t ino</code> <code>dev_t rdev</code>	Place the specified file system object attribute information into the audit record. You can get this information using the <code>stat(2)</code> system call.
	<code>AW_CLEARANCE</code>	

<code>bclear_t *clear</code>	Place the specified clearance into the audit record. If sensitivity labels are not enabled on this system or if the appropriate audit policy (slabel) from <code>auditon(2)</code> is not enabled on this system, the command is ignored.
<code>AW_DATA, char unit_print, char unit_type, char unit_count caddr_t p</code>	Place the specified arbitrary data into the audit record. <i>unit_print</i> describes how the data should be printed by programs that read the audit trail. These are allowable values for <i>unit_print</i> : AWD_BINARY AWD_OCTAL AWD_DECIMAL AWD_HEX AWD_STRING
<i>unit_type</i> describes the type of data in <i>p</i> . These are allowable values for <i>unit_type</i> :	AWD_BYTE AWD_CHAR AWD_SHORT AWD_INT AWD_LONG
<i>unit_count</i> describes how many elements of <i>unit_type</i> exist in <i>p</i> , which is an address pointing to the data to be written.	
<code>AW_EVENT char *event_str</code>	Specify the audit event associated with the audit record. One and only one event must be associated with every audit record. If an attempt is made to write an audit record for which an event has not been specified, <code>auditwrite()</code> returns an error. <i>event_str</i> is any valid user-level audit-event string as defined by <code>audit_event(4)</code> . (<code>AW_EVENT</code> is used for third-party application events. For other events, <code>AW_EVENTNUM</code> should

AW_EVENTNUM

int *event*

AW_EVENTNUM is similar to AW_EVENT but takes be used if possible, since AW_EVENT incurs additional overhead for string lookup.) as its argument any valid *event* number instead of an event string. To maintain compatibility between third party add-ons, only registered events may use this attribute command. *event* is any valid audit event defined in audit_event(4) and </usr/include/bsm/audit_uevent.h>. See audit_event(4) for further information on audit-event strings.

AW_EXEC_ARGS

char ***argv*

Place the specified command line arguments into the audit record. The array is terminated by a null pointer. (The format is the same as that used for *argv* by an invoking C program.) If the appropriate audit policy (*argv*) from auditon(2) is not enabled on this system, this call is ignored.

AW_EXEC_ENV

char ***envp*

Place the specified command-line environment into the audit record. The array is terminated by a null pointer. (This format is the same as that used for *envp* by an invoking C program.) If the appropriate audit policy (*arge*) from auditon(2) is not enabled on this system, this call is ignored.

AW_EXIT, int *status*int *errno*

Place the specified program exit-status information into the audit record. *status* contains the exit status of the calling program. *errno* contains the system error number or an internal error number indicating the cause of the program exit.

AW_GROUPS, int *num*gid_t **groups*

Format and place the elements of the array *groups* into the audit record. *num* specifies the number of elements in the array and must be between NGROUPS_UMIN and NGROUPS_UMAX as defined in </usr/include/sys/param.h>. If the audit policy [see auditconfig(1M)] is not configured for including supplementary groups, the command is ignored.

AW_INADDR

<code>struct in_addr *<i>in_addr</i></code>	Place the specified Internet address into the audit record.
<code>AW_IPC, char <i>type</i>, int <i>id</i></code>	Place the specified interprocess-communications identifier into the audit record. <i>type</i> is one of these values: <code>AT_IPC_MSG</code> , <code>AT_IPC_SEM</code> , <code>AT_IPC_SHM</code> , or <code>AT_IPC_NULL</code> .
<code>AW_IPC_PERM</code> <code>struct ipc_perm *<i>perm</i></code>	Place the specified interprocess-communications identifier permission information into the audit record.
<code>AW_IPORT, u_short <i>ipport</i></code>	Place the specified IP port into the audit record.
<code>AW_LEVEL, blevel_t *<i>level</i></code>	Place the specified level into the audit record. If sensitivity labels are not enabled on this system or if the appropriate audit policy (slabel) from <code>auditon(2)</code> is not enabled on this system, the command is ignored.
<code>AW_OPAQUE, caddr_t <i>data</i></code> <code>short <i>byte_count</i></code>	Place into the audit record the opaque data to which <i>data</i> points and which has <i>byte_count</i> length.
<code>AW_PATH, char *<i>path</i></code>	Place the specified path into the audit record. Paths are anchored with the current active root if they do not begin with a slash (/).
<code>AW_PRIVILEGE,</code> <code>priv_set_t *<i>priv_set</i></code> <code>char <i>settype</i></code>	Place the specified privilege set into the audit record. These are allowable values for <i>settype</i> : <code>AU_PRIV_UNKNOWN</code> <code>AU_PRIV_FORCED</code> <code>AU_PRIV_ALLOWED</code> <code>AU_PRIV_EFFECTIVE</code> <code>AU_PRIV_INHERITABLE</code>

	AU_PRIV_PERMITTED
	AU_PRIV_SAVED
AW_PROCESS, au_id_t <i>audit</i> , uid_t <i>eu</i> id, gid_t <i>eg</i> id uid_t <i>ru</i> id, gid_t <i>rg</i> id, pid_t <i>pi</i> d, au_asid_t <i>si</i> d au_tid_t <i>ti</i> d	<p>Place the specified process information into the audit record. The AW_PROCESS and AW_SUBJECT attributes record the same information. Use the AW_PROCESS attribute when recording information about a process object. Use the AW_SUBJECT attribute when recording information about a process subject.</p>
AW_RETURN, char <i>number</i> u_int <i>retval</i>	<p>Indicates the success or failure of an audit event. This attribute is used by auditwrite() for preselection, and by the auditreduce(1M) post-selection program to select audit records according to success or failure.</p> <p><i>number</i> indicates the success or failure of the event. Failure is denoted by a nonzero <i>number</i> value. Positive values are interpreted by praudit(1M) as errno values. Corresponding error strings are printed. Negative values indicate a general failure specific to the audit event. Success is denoted by a zero <i>number</i> value. <i>retval</i> indicates the return value or status value of the successful or failed function or program.</p>
AW_SLABEL, bslabel_t <i>*label</i>	<p>Place the specified sensitivity label into the audit record. If sensitivity labels are not enabled on this system or if the appropriate audit policy (slabel) from auditon(2) is not enabled on this system, the command is ignored.</p>
AW_SOCKET, struct socket <i>*s</i>	<p>Place the specified socket information into the audit record.</p>

AW_SUBJECT, au_id_t <i>au</i> id, uid_t <i>eu</i> id, gid_t <i>eg</i> id uid_t <i>ru</i> id, gid_t <i>rg</i> id, pid_t <i>pid</i> , au_aside_t <i>sid</i> au_tid_t <i>*tid</i>	Place the specified subject information into the audit record. The AW_PROCESS and AW_SUBJECT attributes record the same information. Use the AW_PROCESS attribute when recording information about a process object. Use the AW_SUBJECT attribute when recording information about a process subject.
AW_TEXT, char <i>*text</i>	Place the specified null-terminated string <i>text</i> into the audit record.
AW_USEOFPRIV, char <i>flag</i> priv_t <i>priv</i>	Place a flag and a single privilege into the audit record denoting an attempted use of privilege. <i>flag</i> indicates success (1) or failure (0) of the attempt. <i>priv</i> indicates the privilege upon which the attempt was made.
AW_XATOM char <i>*atom_string</i>	Place the specified X atom string into the audit record.
AW_XCOLORMAP, u_long <i>xid</i> uid_t <i>creator_uid</i>	Place the specified X colormap information into the audit record.
AW_XCLIENT u_long <i>clientid</i>	Place the specified X client ID into the audit record.
AW_XCURSOR, u_long <i>xid</i> uid_t <i>creator_uid</i>	Place the specified X cursor information into the audit record.
AW_XFONT, u_long <i>xid</i> uid_t <i>creator_uid</i>	Place the specified X font information into the audit record.

AW_XGC, u_long *xid*
 uid_t *creator_uid* Place the specified X gc information into the audit record.

AW_XPIXMAP, u_long
xid
 uid_t *creator_uid* Place the specified X pixel-mapping information into the audit record.

AW_XPROPERTY, u_long
xid, uid_t *creator_uid*,
 char **atom_name* Place the specified X property information into the audit record.

AW_XSELECT, char
 **property_string*, char
 **property_type*,
 char **window_data* Place the specified X select information into the audit record.

AW_XWINDOW, u_long
xid,
 uid_t *creator_uid* Place the specified X window information into the audit record.

Terminator Command

The terminator command **AW_END** must be the last argument on the `auditwrite()` invocation line.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe

RETURN VALUES

`auditwrite()` returns:

- 0 On success.
- 1 On failure, and sets `aw_errno` to indicate the error.

ERRORS

When an error is encountered, any whole or partial audit records are immediately written to the audit trail. These include records that may have been queued. In addition, a **LOG_ALERT** message is sent to `syslogd(1M)` and an attempt is made to write an `auditwrite()` "processing error" audit record to the audit trail.

`aw_strerror(3)` or `aw_perror(3)` may be used for obtaining the error strings associated with `aw_errno`.

These are the possible values of `aw_errno`:

<code>AW_ERR_ADDR_INVALID</code>	An address specified was invalid.
<code>AW_ERR_ALLOC_FAIL</code>	An attempt to allocate memory failed.
<code>AW_ERR_AUDITON_FAIL</code>	The <code>auditon(2)</code> system call failed. See <code>errno</code> for the reason.
<code>AW_ERR_AUDIT_FAIL</code>	The <code>audit(2)</code> system call failed. See <code>errno</code> for the reason.
<code>AW_ERR_CMD_INCOMPLETE</code>	A required command was omitted. This event occurs when <code>AW_APPEND</code> is specified without any attribute commands or vice versa.
<code>AW_ERR_CMD_INVALID</code>	The command specified was not a valid command.
<code>AW_ERR_CMD_IN_EFFECT</code>	A command already in effect, such as <code>AW_QUEUE</code> or <code>AW_SAVERD</code> was specified.
<code>AW_ERR_CMD_NOT_IN_EFFECT</code>	An attempt was made to reverse a command that was not in effect.
<code>AW_ERR_CMD_TOO_MANY</code>	More than one control command was specified.
<code>AW_ERR_EVENT_ID_INVALID</code>	The event ID string passed was not valid.
<code>AW_ERR_EVENT_ID_NOT_SET</code>	An attempt was made to write an audit record without a valid event ID. When this attempt occurs, the event ID is set to a null value and the record is written anyway as a part of error-processing procedure.
<code>AW_ERR_GETAUDIT_FAIL</code>	The <code>getaudit(2)</code> system call failed. See <code>errno</code> for the reason.
<code>AW_ERR_QUEUE_SIZE_INVALID</code>	The specified queue size was greater than the system-imposed maximum audit-record size.
<code>AW_ERR_RD_INVALID</code>	The specified record descriptor was invalid.

AW_ERR_REC_TOO_BIG

An attempt was made to construct an audit record larger than the system-imposed maximum audit record size.

AW_ERR_NO_PLABEL

The process label for the current process could not be obtained; thus a complete record could not be generated.

EXAMPLES

Valid Examples

EXAMPLE 1 Single-shot record construction

```
/* Single-shot record construction:
 * Construct an audit record and write the record to the audit trail.
 * Uses the default audit record.
 */
```

```
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string1, AW_TEXT, "hello", AW_WRITE, AW_END);
```

EXAMPLE 2 Multi-shot construction

```
/* Multi-shot construction:
 * Construct an audit record piecemeal and write the record.
 * Uses the default audit record.
 */
```

```
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string2, AW_APPEND, AW_END);
(void) auditwrite(AW_TEXT, "part 1", AW_APPEND, AW_END);
(void) auditwrite(AW_TEXT, "part 2", AW_APPEND, AW_END);
(void) auditwrite(AW_RETURN, 0, 0, AW_APPEND, AW_END);
(void) auditwrite(AW_WRITE, AW_END);
```

EXAMPLE 3 Multi-shot record construction

```
/* Multi-shot record construction:
 * Decide upon the return token value when it occurs.
 */
```

```
(void) auditwrite(AW_EVENTNUM, AUE_ftpd, AW_APPEND, AW_END);
(void) auditwrite(AW_TEXT, "Read access attempt", AW_APPEND, AW_END);
```

```
if (access_decision() == FALSE) {
    succ_or_fail = -1;
    reason = get_reason;
} else {
    succ_or_fail = 0;
    reason = 0;
}
```

```
(void) auditwrite(AW_TEXT, "more text", AW_RETURN, succ_or_fail, reason, AW_APPEND, AW_END);
(void) auditwrite(AW_WRITE, AW_END);
```

EXAMPLE 4 Queueing

```
/* Queueing:
 * Turn on queueing, queue two records, then turn off queueing. Queue is flushed
 * automatically when queueing is turned off.
 */
```

```
(void) auditwrite(AW_QUEUE, 1024, AW_END);
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string3, AW_RETURN, 0, 0, AW_WRITE, AW_END);
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string4, AW_RETURN, 0, 0, AW_WRITE, AW_END);
(void) auditwrite(AW_EVENTNUM, AUE_valid_event_string5, AW_RETURN, 0, 0, AW_APPEND, AW_END);
(void) auditwrite(AW_NOQUEUE, AW_END);
```

Invalid Examples**EXAMPLE 5** Only one control command may be specified

```
/*
 * Invalid command combinations:
 * Only one control command may be specified.
 */
(void) auditwrite(AW_EVENTNUM, valid_event_str, AW_TEXT, "text", AW_DISCARD, AW_WRITE, AW_END);
```

EXAMPLE 6 No control command specified

```
/*
 * Invalid command combinations:
 * No control command specified
 */
(void) auditwrite(AW_TEXT, "text", AW_END);
```

FILES

/etc/security/audit_event	Used to obtain the mappings between audit event strings and audit event numbers.
---------------------------	--

SEE ALSO

Trusted Solaris 7
Reference Manual

auditreduce(1M), praudit(1M), audit(2), getaudit(2), stat(2),
audit_event(4), auditconfig(1M), auditon(2)

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

When a subject is not provided, `auditwrite()` attempts to generate the subject, groups, and (depending on appropriate audit policy) sensitivity label attributes for the current process. This attempt has implications for servers because unless they negotiate to get the AUID, UID, sensitivity label, and groups of the process being served and provide them to `auditwrite()`, the values recorded will be those of the server process. Programmers of servers should take this circumstance into account when using `auditwrite()` and make specific requests to `auditwrite()` to work around this problem.

To write an audit record with an event number from 2048 to 32767, the calling process must have `PRIV_PROC_AUDIT_TCB` in its set of effective privileges. If the event number is from 32768 to 65535, the calling process must have `PRIV_PROC_AUDIT_APPL` in its set of effective privileges. These sets of event numbers are the only valid user-level event numbers.

Information labels (ILs) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any ILs on communications and files from systems running earlier releases as `ADMIN_LOW`.

- Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- ILs do not display in window labels; SLs (Sensitivity Labels) display alone within brackets.
- ILs do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting ILs are always `ADMIN_LOW`, and cannot be set on any objects.
- In auditing, the `ilabel` token is recorded as `ADMIN_LOW`, when it is recorded. The audit event numbers 519 (`AUE_OFLOAT`), 520 (`AUE_SFLOAT`), and 9036 (`AUE_iil_change`) continue to be reserved, but those events are no longer recorded.

NAME	au_preselect – Preselect an audit event										
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...]</pre>										
DESCRIPTION	<pre>#include <bsm/libbsm.h> int au_preselect(au_event_t event, au_mask_t *mask_p, int sorf, int flag);</pre> <p><code>au_preselect()</code> determines whether or not the audit event <i>event</i> is preselected against the binary preselection mask pointed to by <i>mask_p</i> (usually obtained by a call to <code>getaudit(2)</code>). <code>au_preselect()</code> looks up the classes associated with <i>event</i> in <code>audit_event(4)</code> and compares them with the classes in <i>mask_p</i>. If the classes associated with <i>event</i> match the classes in the specified portions of the binary preselection mask pointed to by <i>mask_p</i>, the event is said to be preselected.</p> <p><i>sorf</i> indicates whether the comparison is made with the success portion, the failure portion or both portions of the mask pointed to by <i>mask_p</i>.</p> <p>The following are the valid values of <i>sorf</i>:</p> <table> <tr> <td>AU_PRS_SUCCESS</td><td>Compare the event class with the success portion of the preselection mask.</td></tr> <tr> <td>AU_PRS_FAILURE</td><td>Compare the event class with the failure portion of the preselection mask.</td></tr> <tr> <td>AU_PRS_BOTH</td><td>Compare the event class with both the success and failure portions of the preselection mask.</td></tr> </table> <p><i>flag</i> tells <code>au_preselect()</code> how to read the <code>audit_event(4)</code> database. Upon initial invocation, <code>au_preselect()</code> reads the <code>audit_event(4)</code> database and allocates space in an internal cache for each entry with <code>malloc(3C)</code>. In subsequent invocations, the value of <i>flag</i> determines where <code>au_preselect()</code> obtains audit event information. The following are the valid values of <i>flag</i>:</p> <table> <tr> <td>AU_PRS_REREAD</td><td>Get audit event information by searching the <code>audit_event(4)</code> database.</td></tr> <tr> <td>AU_PRS_USECACHE</td><td>Get audit event information from internal cache created upon the initial invocation. This option is much faster.</td></tr> </table>	AU_PRS_SUCCESS	Compare the event class with the success portion of the preselection mask.	AU_PRS_FAILURE	Compare the event class with the failure portion of the preselection mask.	AU_PRS_BOTH	Compare the event class with both the success and failure portions of the preselection mask.	AU_PRS_REREAD	Get audit event information by searching the <code>audit_event(4)</code> database.	AU_PRS_USECACHE	Get audit event information from internal cache created upon the initial invocation. This option is much faster.
AU_PRS_SUCCESS	Compare the event class with the success portion of the preselection mask.										
AU_PRS_FAILURE	Compare the event class with the failure portion of the preselection mask.										
AU_PRS_BOTH	Compare the event class with both the success and failure portions of the preselection mask.										
AU_PRS_REREAD	Get audit event information by searching the <code>audit_event(4)</code> database.										
AU_PRS_USECACHE	Get audit event information from internal cache created upon the initial invocation. This option is much faster.										
RETURN VALUES	<p><code>au_preselect()</code> returns:</p> <table> <tr> <td>0</td><td><i>event</i> is not preselected.</td></tr> <tr> <td>1</td><td><i>event</i> is preselected.</td></tr> <tr> <td>-1</td><td>An error occurred. <code>au_preselect()</code> couldn't allocate memory or couldn't find <i>event</i> in the <code>audit_event(4)</code> database.</td></tr> </table>	0	<i>event</i> is not preselected.	1	<i>event</i> is preselected.	-1	An error occurred. <code>au_preselect()</code> couldn't allocate memory or couldn't find <i>event</i> in the <code>audit_event(4)</code> database.				
0	<i>event</i> is not preselected.										
1	<i>event</i> is preselected.										
-1	An error occurred. <code>au_preselect()</code> couldn't allocate memory or couldn't find <i>event</i> in the <code>audit_event(4)</code> database.										

SUMMARY OF TRUSTED SOLARIS CHANGES

This function looks up the classes associated with *event* in `audit_event(4)` rather than in `audit_event(4)`. By default, auditing is enabled in the Trusted Solaris environment.

FILES

<code>/etc/security/audit_class</code>	Maps audit class number to audit class names and descriptions.
<code>/etc/security/audit_event</code>	Maps audit event number to audit event names.

ATTRIBUTES

See `attributes(5)` for a description of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`getaudit(2)`, `auditwrite(3)`, `getauclassent(3)`, `getauevent(3)`,
`audit_class(4)`, `audit_event(4)`

**SunOS 5.7 Reference
Manual**

`malloc(3C)`, `attributes(5)`

NOTES

This functionality is active only if the auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration* for how to disable and enable auditing. `au_preselect()` is normally called prior to constructing and writing an audit record. If the event is not preselected, the overhead of constructing and writing the record can be saved.

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – Translate and verify user authorizations
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id); char auth_set_to_str(auth_set_t * authset, char separator); auth_t str_to_auth(char * auth_name); auth_set_t *str_to_auth_set(char * auth_names, char * separators); char *get_auth_text(auth_t auth_id); int chkauth(auth_t auth_id, char * username); void free_auth_set(auth_set_t * authset);</pre>
DESCRIPTION	<p>auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i> . If <i>auth_id</i> is an undefined authorization ID , the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID , TSOL_MAX_AUTH , a NULL is returned.</p> <p>auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C) . Authorization names are separated by the character <i>separator</i> . Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list.</p> <p>str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i> . <i>auth_name</i> is the name in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH .</p> <p>str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i> . <i>auth_names</i> are names in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . The string none is translated to an empty authorization list (NULL) . This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization</p>

RETURN VALUES

names appear in the *auth_names* string, the function converts these invalid authorizations into *auth_id*s with the value of 0 .

get_auth_text() returns a pointer to the statically-allocated, null-terminated authorization description text specified by *auth_id* .

chkauth() returns 1 if the user, specified by *username* has the authorization specified by *auth_id* .

free_auth_set() releases the memory returned by *str_to_auth_set*() .

auth_to_str() returns a pointer to the translated authorization name string. It returns NULL on failure.

str_to_auth() returns the numeric authorization id. It returns 0 on failure.

auth_set_to_str() returns a pointer to the translated authorization names string. It returns NULL on failure.

str_to_auth_set() returns NULL on success.

get_auth_text() return a string on success and NULL upon failure.

chkauth() returns 1 on success and 0 on failure.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

NOTES

This interface is uncommitted, which means it may change between minor releases of the Trusted Solaris environment. To use these routines, the program must be loaded with the Trusted Solaris library *libtsolddb* .

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auth_desc(4) , *auth_name*(4)

attributes(5)

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – Translate and verify user authorizations
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id); char auth_set_to_str(auth_set_t * authset, char separator); auth_t str_to_auth(char * auth_name); auth_set_t *str_to_auth_set(char * auth_names, char * separators); char *get_auth_text(auth_t auth_id); int chkauth(auth_t auth_id, char * username); void free_auth_set(auth_set_t * authset);</pre>
DESCRIPTION	<p>auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i> . If <i>auth_id</i> is an undefined authorization ID , the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID , TSOL_MAX_AUTH , a NULL is returned.</p> <p>auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C) . Authorization names are separated by the character <i>separator</i> . Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list.</p> <p>str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i> . <i>auth_name</i> is the name in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH .</p> <p>str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i> . <i>auth_names</i> are names in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . The string none is translated to an empty authorization list (NULL) . This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization</p>

names appear in the *auth_names* string, the function converts these invalid authorizations into *auth_id*s with the value of 0 .

get_auth_text() returns a pointer to the statically-allocated, null-terminated authorization description text specified by *auth_id* .

chkauth() returns 1 if the user, specified by *username* has the authorization specified by *auth_id* .

free_auth_set() releases the memory returned by *str_to_auth_set*() .

RETURN VALUES

auth_to_str() returns a pointer to the translated authorization name string. It returns NULL on failure.

str_to_auth() returns the numeric authorization id. It returns 0 on failure.

auth_set_to_str() returns a pointer to the translated authorization names string. It returns NULL on failure.

str_to_auth_set() returns NULL on success.

get_auth_text() return a string on success and NULL upon failure.

chkauth() returns 1 on success and 0 on failure.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

NOTES

This interface is uncommitted, which means it may change between minor releases of the Trusted Solaris environment. To use these routines, the program must be loaded with the Trusted Solaris library *libtsolddb* .

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auth_desc(4) , *auth_name*(4)

attributes(5)

NAME	au_user_mask – Get user's binary preselection mask				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int au_user_mask(char *username, au_mask_t *mask_p);</pre>				
DESCRIPTION	<p>au_user_mask() reads the default, system wide audit classes from audit_control(4), combines them with the per-user audit classes from the audit_user(4) database, and updates the binary preselection mask pointed to by <i>mask_p</i> with the combined value.</p> <p>The audit flags in the <i>flags</i> field of the audit_control(4) database and the <i>always-audit-flags</i> and <i>never-audit-flags</i> from the audit_user(4) database represent binary audit classes. These fields are combined by au_preselect(3) as follows:</p> $\text{mask} = (\text{flags} + \text{always-audit-flags}) - \text{never-audit-flags}$ <p>au_user_mask() only fails if both the both the audit_control(4) and the audit_user(4) database entries could not be retrieved. This allows for flexible configurations.</p>				
RETURN VALUES	<p>au_user_mask() returns:</p> <p>0 Success.</p> <p>-1 Failure. Both the audit_control(4) and the audit_user(4) database entries could not be retrieved.</p>				
FILES	<p>/etc/security/audit_control Contains default parameters read by the audit daemon, auditd(1M).</p> <p>/etc/security/audit_user Stores per-user audit event mask.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <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>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				
SUMMARY OF TRUSTED SOLARIS CHANGES	By default, auditing is enabled in the Trusted Solaris environment. Trusted Solaris 2.5.1 and 7 extend the number of audit classes and audit events, and introduce new but similar structures and programming interfaces.				
SEE ALSO					

Trusted Solaris 7 Reference Manual	login(4), getaudit(4), au_preselect(3), getacinfo(3), getauusernam(3), audit_control(4), audit_user(4)
SunOS 5.7 Reference Manual	attributes(5)
NOTES	This functionality is active only if auditing has been enabled. au_user_mask() should be called by programs like login(1) that set the preselection mask of a process with setaudit(2) in the Trusted Solaris 7 Reference Manual. getaudit(2) should be used to obtain audit characteristics for the current process.

NAME aw_strerror, aw_errno, aw_perror – Obtain and display error messages

SYNOPSIS `cc [flag...] file... -lbsm -lsocket -lnsl -lintl -ltsol [library...]`

```
#include <bsm/auditwrite.h>
int aw_errno;
char *aw_strerror(const int status);

void aw_perror(const char *s);
```

DESCRIPTION aw_errno is a variable set by auditwrite(3) to indicate the type of error encountered during its execution, in much the same manner as errno(3C) is set during a system or library call. The supporting functions convert auditwrite(3) error numbers into text strings.

aw_strerror() takes a specified return value *status* and returns a pointer to a string constant that is the error string.

aw_perror() prints the error message corresponding to *status* as " *s* : *error_message* " to standard error.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auditwrite(3)

errno(3C) , perror(3C) , strerror(3C)

NOTES

The returned string must not be overwritten. If string manipulation is required, work on a local copy.

NAME	aw_strerror, aw_errno, aw_perror – Obtain and display error messages						
SYNOPSIS	<pre>cc [flag...] file... -lbsm -lsocket -lnsl -lintl -ltsol [library...]</pre> <pre>#include <bsm/auditwrite.h> int aw_errno; char * aw_strerror(const int status); void aw_perror(const char * s);</pre>						
DESCRIPTION	<p>aw_errno is a variable set by auditwrite(3) to indicate the type of error encountered during its execution, in much the same manner as errno(3C) is set during a system or library call. The supporting functions convert auditwrite(3) error numbers into text strings.</p> <p>aw_strerror() takes a specified return value <i>status</i> and returns a pointer to a string constant that is the error string.</p> <p>aw_perror() prints the error message corresponding to <i>status</i> as " <i>s</i> : <i>error_message</i> " to standard error.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWcsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWcsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWcsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	auditwrite(3)						
SunOS 5.7 Reference Manual	errno(3C) , perror(3C) , strerror(3C)						
NOTES	The returned string must not be overwritten. If string manipulation is required, work on a local copy.						

NAME aw_strerror, aw_errno, aw_perror – Obtain and display error messages

SYNOPSIS `cc [flag...] file... -lbsm -lsocket -lnsl -lintl -ltsol [library...]`

```
#include <bsm/auditwrite.h>
int aw_errno;
char *aw_strerror(const int status);

void aw_perror(const char *s);
```

DESCRIPTION aw_errno is a variable set by auditwrite(3) to indicate the type of error encountered during its execution, in much the same manner as errno(3C) is set during a system or library call. The supporting functions convert auditwrite(3) error numbers into text strings.

aw_strerror() takes a specified return value *status* and returns a pointer to a string constant that is the error string.

aw_perror() prints the error message corresponding to *status* as " *s* : *error_message* " to standard error.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auditwrite(3)

errno(3C) , perror(3C) , strerror(3C)

NOTES

The returned string must not be overwritten. If string manipulation is required, work on a local copy.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bsllabel_t * <i>label</i>); void bsllhigh(bsllabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bsllabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bsllabel_t * <i>label</i>); void bsllhigh(bsllabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bsllabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	btohex, bcltoh, bsltoh, biltoh, bcleartoh, bcltoh_r, bsltoh_r, biltoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char * bcltoh(const bclabel_t * label); char * bsltoh(const bslabel_t * label); char * bcleartoh(const bclear_t * clearance); char * bcltoh_r(const bclabel_t * label, char * hex); char * bsltoh_r(const bslabel_t * label, char * hex); char * bcleartoh_r(const bclear_t * clearance, char * hex); char * h_alloc(const unsigned char type); void h_free(char * hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), biltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

`h_free()` 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

`atohexlabel(1M)`, `hextoalabel(1M)`, `bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltos(3)`, `bltype(3)`, `blvalid(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

The functions `bcltoh()`, `bsltoh()`, `biltoh()`, and `bcleartoh()` 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.

For multithreaded applications, the functions `bcltoh_r()`, `bsltoh_r()`, `biltoh_r()`, and `bcleartoh_r()` should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	btohex, bcltoh, bsloth, bilttoh, bclearth, bcltoh_r, bsloth_r, bilttoh_r, bclearth_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char *bcltoh(const bclabel_t *label); char *bsloth(const bslabel_t *label); char *bclearth(const bclear_t *clearance); char *bcltoh_r(const bclabel_t *label, char *hex); char *bsloth_r(const bslabel_t *label, char *hex); char *bclearth_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsloth() and bsloth_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bclearth() and bclearth_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsloth_r(), bilttoh_r(), and bclearth_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

`h_free()` 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

`atohexlabel(1M)`, `hextoalabel(1M)`, `bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltos(3)`, `bltype(3)`, `blvalid(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

The functions `bcltoh()`, `bsltoh()`, `biltoh()`, and `bcleartoh()` 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.

For multithreaded applications, the functions `bcltoh_r()`, `bsltoh_r()`, `biltoh_r()`, and `bcleartoh_r()` should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW.

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL]; however, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	bltos, bcltos, bsltos, biltos, bcleartos – Translate binary labels to character coded labels				
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> int bltos(const blevel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bsltos(const bslabel_t * label, char ** string, const int str_len, const int flags); int bcleartos(const bclear_t * label, char ** string, const int str_len, const int flags);</pre>				
DESCRIPTION	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <p><i>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</i></p> <p>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.</p> <p><i>string</i> may point to either a pointer to pre-allocated memory, or the value (char *) 0 . If it points to a pointer to pre-allocated memory, then <i>str_len</i> indicates the size of that memory. If it points to the value (char *) 0 , memory is allocated using malloc() to contain the translated character-coded labels. The translated <i>label</i> is copied into allocated or pre-allocated memory.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i> .</td></tr> <tr> <td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr> </table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.
LONG_WORDS	Translate using long names of words defined in <i>label</i> .				
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.				

LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the label_encodings file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name 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.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

INFORMATION LABEL [SENSITIVITY LABEL]

The applicable *flags* are LONG_WORDS or SHORT_WORDS , and VIEW_EXTERNAL or VIEW_INTERNAL . A *flags* value 0 is equivalent to (LONG_WORDS).

bsltos() 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).

biltos() translates a binary information label into a string. The applicable *flags* are LONG_CLASSIFICATION or SHORT_CLASSIFICATION , LONG_WORDS or SHORT_WORDS , ALL_ENTRIES or ACCESS_RELATED , VIEW_EXTERNAL or VIEW_INTERNAL , and NO_CLASSIFICATION . A *flags* value 0 is equivalent to (LONG_CLASSIFICATION | LONG_WORDS | ALL_ENTRIES).

bcleartos() 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 may not be the same as the translation of a sensitivity label. These functions

use different `label_encodings` file tables that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These routines return:

- 1 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 the `label_encodings` file is inaccessible.
- 0 If memory cannot be allocated for the return string, or the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the NULL string (`*string[0]='\00'`).
- >0 If successful, the length of the character-coded label including the NULL terminator.

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.

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.

SEE ALSO

Trusted Solaris 7
Reference Manual

`bcltobanner(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltype(3)`, `blvalid(3)`, `btohex(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`, `label_encodings(4)`

Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual****NOTES**

`free(3C)`, `malloc(3C)`, `attributes(5)`

If memory is allocated by these routines, the caller must free the memory with `free()` when the memory is no longer in use.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bsllabel_t * <i>label</i>); void bsllhigh(bsllabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bsllabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blvalid, bslvalid, bilvalid, bclearvalid – Check validity of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h> int bslvalid(const bslabel_t * label); int bclearvalid(const bclear_t * clearance);</pre>						
DESCRIPTION	<p>The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current process' sensitivity label.</p> <p>These functions check the validity of binary labels.</p> <p>bslvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system.</p> <p>bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <p>-1 If the label_encodings file is inaccessible.</p> <p>0 If the binary label is not valid for this system or is not dominated by the process' sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges,</p> <p>1 If the binary label is valid for this system.</p>						
FILES	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
SEE ALSO	<p>bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) , blmanifest(3) , blminmax(3) , blportion(3) , bltocolour(3) , bltos(3) , bltype(3) , btohex(3) , hextob(3) , labelinfo(3) , labelvers(3) , sbltos(3) , stobl(3) , label_encodings(4)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Binary sensitivity labels are *valid* if they are contained in the `SYSTEM_ACCREDITATION_RANGE` as checked by `blinset(3)`. `bslvalid()` is a synonym for calling `blinset()` with the containing set of `SYSTEM_ACCREDITATION_RANGE` and is included for completeness.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bsllabel_t * <i>label</i>); void bsllhigh(bsllabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bsllabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bsllabel_t * <i>label</i>); void bsllhigh(bsllabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bsllabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	bcltobanner – Translate binary CMW labels to character-coded labels for a printer banner page
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]</pre>
DESCRIPTION	<pre>#include <tsol/label.h> int bcltobanner(const bclabel_t *label, struct banner_fields *fields, const int flags);</pre> <p>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' sensitivity label.</p> <p>bcltobanner() translates a binary CMW label, <i>label</i>, into various character-coded labels and strings for display on printer banner and trailer pages and at the top and bottom of the document body pages. The members of the <i>fields</i> structure are either string pointers, or the length of memory pre-allocated to a string pointer. The string pointers may contain either a pointer to pre-allocated memory, or the value (char *)0. If the string pointer contains a pointer to pre-allocated memory, then its associated length member indicates the size of that memory. If it contains the value (char *)0, memory is allocated using malloc() to contain the translated character-coded label or string. The translated string is copied into allocated or pre-allocated memory.</p> <p>The structure banner_fields stores the folloiwng information:</p> <pre>struct banner_fields { char *header; /* top and bottom banner/trailer page */ char *protect_as; /* "protect as" banner page section */ char *ilabel; /* Information Label, also top and bottom of body pages */ char *caveats; /* ``caveats`` banner page section */ char *channels; /* ``handling channels`` section */ /* lengths of pre-allocated string memory */ short header_len; /* header */ short protect_as_len; /* protect_as */ short short ilabel_len; /* Information Label */ short caveats_len; /* caveats */ short channels_len; /* handling channels */ };</pre> <p>Members of the <i>fields</i> structure have the following meaning:</p> <p>header String to print at the top and bottom of banner and trailer pages</p> <p>protect_as String to print in the protect as warning of banner and trailer pages</p> <p>ilabel String to print in the this system has labeled message of banner and trailer pages, and at the top and bottom of the document body pages</p> <p>caveats</p>

String to print in `caveats` section of the banner and trailer pages

`channels`

String to print in `channels` section of the banner and trailer pages

`header_len`

Length of pre-allocated memory for `header` string

`protect_as_len`

Length of pre-allocated memory for `protect_as` string

`ilabel_len`

Length of pre-allocated memory for `ilabel` string

`caveats_len`

Length of pre-allocated memory for `caveats` string

`channels_len`

Length of pre-allocated memory for `channels` string

The translation is controlled by the value of the *flags* parameter. *flags* may be either `LONG_WORDS`, `SHORT_WORDS`, or 0 (zero).

`LONG_WORDS` Translate using long names of words defined in *label*.

`SHORT_WORDS` 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.

0 A *flags* value 0 is equivalent to `LONG_WORDS`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

`bcltobanner()` returns:

-1 If *label* is not a binary CMW label with a defined information label and sensitivity label, or if its information label portion is not dominated by its sensitivity label portion, or if its sensitivity label portion 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.

0 If memory cannot be allocated for a string in the *fields* structure, or if one of the pre-allocated memories is insufficient to hold its string. The

value of that pre-allocated string is set to the NULL string (*fields*→*string*[0] = '\00';).

1 If successful.

EXAMPLES

EXAMPLE 1 Banner Page Format

The string members of the *fields* structure are included in a printer banner page in the following manner:

HEADER

This output must be protected as:

PROTECT_AS

unless manually reviewed and downgraded.

The system has labeled this data:

ILABEL
CAVEATS
CHANNELS
HEADER

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.

SEE ALSO

Trusted Solaris 7
Reference Manual

bilconjoin(3), *blcompare*(3), *blinset*(3), *blmanifest*(3), *blminmax*(3),
blportion(3), *bltos*(3), *bltype*(3), *blvalid*(3), *btohex*(3), *hextob*(3),
labelinfo(3), *labelvers*(3), *sbltos*(3), *stobl*(3), *label_encodings*(4)

Trusted Solaris Developer's Guide, and *Trusted Solaris administrator's document set*

SunOS 5.7 Reference
Manual

free(3C), *malloc*(3C)

NOTES

If memory is allocated by this routine, the caller must free memory with *free*() when the memory is no longer in use. ADMIN_LOW and ADMIN_HIGH label portions are mapped differently than the other routines. If the sensitivity label portion of *label* is ADMIN_LOW, the *label* is mapped into the minimum sensitivity label defined in the *label_encodings* file, and if the *label* is ADMIN_HIGH, the *label* is mapped into the maximum classification, and all compartments defined in the *label_encodings* file.

Information labels (ILs) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any ILs on communications and files from systems running earlier releases as ADMIN_LOW.

- Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.
As a result, Trusted Solaris 7 has the following characteristics:
 - ILs do not display in window labels; SLs (Sensitivity Labels) display alone within brackets.
 - ILs do not float.
 - Setting an IL on an object has no effect.
 - Getting an object's IL will always return `ADMIN_LOW`.
 - Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting ILs are always `ADMIN_LOW`, and cannot be set on any objects.
 - Printer banners display sensitivity labels, not information labels.

NAME	btohex, bcltoh, bsltoh, bilttoh, bcleartoh, bcltoh_r, bsltoh_r, bilttoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char * bcltoh(const bclabel_t * <i>label</i>); char * bsltoh(const bslabel_t * <i>label</i>); char * bcleartoh(const bclear_t * <i>clearance</i>); char * bcltoh_r(const bclabel_t * <i>label</i> , char * <i>hex</i>); char * bsltoh_r(const bslabel_t * <i>label</i> , char * <i>hex</i>); char * bcleartoh_r(const bclear_t * <i>clearance</i> , char * <i>hex</i>); char * h_alloc(const unsigned char <i>type</i>); void h_free(char * <i>hex</i>);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p><i>0x information_label_hexadecimal_value [0x sensitivity_label_hexadecimal_value]</i></p> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <p><i>[0x sensitivity_label_hexadecimal_value]</i></p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p><i>0x clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), bilttoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

`h_free()` 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

`atohexlabel(1M)`, `hextoalabel(1M)`, `bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltocolor(3)`, `bltos(3)`, `bltype(3)`, `blvalid(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

The functions `bcltoh()`, `bsltoh()`, `biltoh()`, and `bcleartoh()` 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.

For multithreaded applications, the functions `bcltoh_r()`, `bsltoh_r()`, `biltoh_r()`, and `bcleartoh_r()` should be used.

Information labels (ILs) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any ILs on communications and files from systems running earlier releases as ADMIN_LOW.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 has the following characteristics:

- ILs do not display in window labels; SLs (Sensitivity Labels) display alone within brackets.
- ILs do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	btohex, bcltoh, bslttoh, bilttoh, bcleartoh, bcltoh_r, bslttoh_r, bilttoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char *bcltoh(const bclabel_t *label); char *bslttoh(const bslabel_t *label); char *bcleartoh(const bclear_t *clearance); char *bcltoh_r(const bclabel_t *label, char *hex); char *bslttoh_r(const bslabel_t *label, char *hex); char *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bslttoh() and bslttoh_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bslttoh_r(), bilttoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

`h_free()` 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

`atohexlabel(1M)`, `hextoalabel(1M)`, `bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltos(3)`, `bltype(3)`, `blvalid(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

The functions `bcltoh()`, `bsltoh()`, `biltoh()`, and `bcleartoh()` 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.

For multithreaded applications, the functions `bcltoh_r()`, `bsltoh_r()`, `biltoh_r()`, and `bcleartoh_r()` should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW.

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL]; however, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	blportion, bcltosl, bcltoil, biltollev, getcsl, getcil, setcsl, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> bslabel_t * bcltosl(bslabel_t * label); void getcsl(bslabel_t * destination_label, const bslabel_t * source_label); void setcsl(bslabel_t * destination_label, const bslabel_t * source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() and bcltoil() provide a pointer to the sensitivity label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcsl() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcsl() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blegal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	bltos, bcltos, bsltos, biltos, bcleartos – Translate binary labels to character coded labels				
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> int bltos(const blevel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bsltos(const bslabel_t * label, char ** string, const int str_len, const int flags); int bcleartos(const bclear_t * label, char ** string, const int str_len, const int flags);</pre>				
DESCRIPTION	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <p><i>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</i></p> <p>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.</p> <p><i>string</i> may point to either a pointer to pre-allocated memory, or the value (char *) 0 . If it points to a pointer to pre-allocated memory, then <i>str_len</i> indicates the size of that memory. If it points to the value (char *) 0 , memory is allocated using malloc() to contain the translated character-coded labels. The translated <i>label</i> is copied into allocated or pre-allocated memory.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i> .</td></tr> <tr> <td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr> </table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.
LONG_WORDS	Translate using long names of words defined in <i>label</i> .				
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.				

LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the label_encodings file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name 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.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

INFORMATION LABEL [SENSITIVITY LABEL]

The applicable *flags* are LONG_WORDS or SHORT_WORDS , and VIEW_EXTERNAL or VIEW_INTERNAL . A *flags* value 0 is equivalent to (LONG_WORDS).

bsltos() 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).

biltos() translates a binary information label into a string. The applicable *flags* are LONG_CLASSIFICATION or SHORT_CLASSIFICATION , LONG_WORDS or SHORT_WORDS , ALL_ENTRIES or ACCESS_RELATED , VIEW_EXTERNAL or VIEW_INTERNAL , and NO_CLASSIFICATION . A *flags* value 0 is equivalent to (LONG_CLASSIFICATION | LONG_WORDS | ALL_ENTRIES).

bcleartos() 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 may not be the same as the translation of a sensitivity label. These functions

use different `label_encodings` file tables that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These routines return:

- 1 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 the `label_encodings` file is inaccessible.
- 0 If memory cannot be allocated for the return string, or the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the NULL string (`*string[0]='\00'`).
- >0 If successful, the length of the character-coded label including the NULL terminator.

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.

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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`bcltobanner(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltype(3)`, `blvalid(3)`, `btohex(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`, `label_encodings(4)`

Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual****NOTES**

`free(3C)`, `malloc(3C)`, `attributes(5)`

If memory is allocated by these routines, the caller must free the memory with `free()` when the memory is no longer in use.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	blportion, bcltosl, bcltoil, biltollev, getcsl, getcil, setcsl, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> bslabel_t * bcltosl(bslabel_t * label); void getcsl(bslabel_t * destination_label, const bslabel_t * source_label); void setcsl(bslabel_t * destination_label, const bslabel_t * source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() and bcltoil() provide a pointer to the sensitivity label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcsl() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcsl() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blegal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), bltocolour(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

`attributes(5)`

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW` .

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]` ; however, the IL component is fixed at `ADMIN_LOW` .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bsllabel_t * <i>label</i>); void bsllhigh(bsllabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bsllabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	bilconjoin – Conjoin binary information labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>void bilconjoin(bilabel_t *receiving_label, const bilabel_t *adding_label);</pre>						
DESCRIPTION	bilconjoin() replaces the contents of binary information label <i>receiving_label</i> with the conjunction of binary information labels <i>receiving_label</i> and <i>adding_label</i> .						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO Trusted Solaris 7 Reference Manual SunOS 5.7 Reference Manual	bcltobanner(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3) attributes(5) <i>Trusted Solaris Developer's Guide</i>						

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> .</p> <p>bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> .</p> <p>bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * label); void bclhigh(bclabel_t * label); void bsllow(bslabel_t * label); void bsllhigh(bslabel_t * label); void bclearlow(bclear_t * clearance); void bclearhigh(bclear_t * clearance); void bclundef(bclabel_t * label); void bsllundef(bslabel_t * label); void bclearundef(bclabel_t * label);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef() , bsllundef() , and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * label); void bclhigh(bclabel_t * label); void bsllow(bslabel_t * label); void bsllhigh(bslabel_t * label); void bclearlow(bclear_t * clearance); void bclearhigh(bclear_t * clearance); void bclundef(bclabel_t * label); void bsllundef(bslabel_t * label); void bclearundef(bclabel_t * label);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef() , bsllundef() , and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hex tob(3), labelinfo(3), labelvers(3), sb ltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	btohex, bcltoh, bsloth, bilton, bcleartoh, bcltoh_r, bsloth_r, bilton_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char *bcltoh(const bclabel_t *label); char *bsloth(const bslabel_t *label); char *bcleartoh(const bclear_t *clearance); char *bcltoh_r(const bclabel_t *label, char *hex); char *bsloth_r(const bslabel_t *label, char *hex); char *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsloth() and bsloth_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsloth_r(), bilton_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

`h_free()` 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

`atohexlabel(1M)`, `hextoalabel(1M)`, `bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltos(3)`, `bltype(3)`, `blvalid(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

The functions `bcltoh()`, `bsltoh()`, `biltoh()`, and `bcleartoh()` 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.

For multithreaded applications, the functions `bcltoh_r()`, `bsltoh_r()`, `biltoh_r()`, and `bcleartoh_r()` should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	btohex, bcltoh, bsloth, biltoh, bcleartoh, bcltoh_r, bsloth_r, biltoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char * bcltoh(const bclabel_t * label); char * bsloth(const bslabel_t * label); char * bcleartoh(const bclear_t * clearance); char * bcltoh_r(const bclabel_t * label, char * hex); char * bsloth_r(const bslabel_t * label, char * hex); char * bcleartoh_r(const bclear_t * clearance, char * hex); char * h_alloc(const unsigned char type); void h_free(char * hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsloth() and bsloth_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsloth_r(), biltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

h_free() 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

atoxhexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltocolor(3), bltos(3), bltype(3), blvalid(3), hextob(3), labelinfo(3), labelvers(3), sblltos(3), stobl(3)

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

The functions bclttoh(), bslttoh(), bilstoh(), and bcleartoh() 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.

For multithreaded applications, the functions bclttoh_r(), bslttoh_r(), bilstoh_r(), and bcleartoh_r() should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	blportion, bcltosl, bcltoil, biltolev, getcsl, getcil, setcsl, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> bslabel_t * bcltosl(bclabel_t * <i>label</i>);</pre> <pre>void getcsl(bslabel_t * <i>destination_label</i> , const bclabel_t * <i>source_label</i>);</pre> <pre>void setcsl(bclabel_t * <i>destination_label</i> , const bslabel_t * <i>source_label</i>);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() and bcltoil() provide a pointer to the sensitivity Label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcsl() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcsl() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blequal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>Trusted Solaris 7 Reference Manual</p> <p>bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) , blmanifest(3) , blminmax(3) , bltocolor(3) , bltos(3) , bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) , labelvers(3) , sbltos(3) , stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	bltos, bcltos, bsltos, biltos, bcleartos – Translate binary labels to character coded labels				
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h></pre> <pre>int bltos(const blevel_t * label, char ** string, const int str_len, const int flags);</pre> <pre>int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags);</pre> <pre>int bsltos(const bslabel_t * label, char ** string, const int str_len, const int flags);</pre> <pre>int bcleartos(const bclear_t * label, char ** string, const int str_len, const int flags);</pre>				
DESCRIPTION	<p>The calling process must have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges to perform label translation on labels that dominate the current process' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <p><i>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</i></p> <p>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.</p> <p><i>string</i> may point to either a pointer to pre-allocated memory, or the value <code>(char *)0</code>. If it points to a pointer to pre-allocated memory, then <i>str_len</i> indicates the size of that memory. If it points to the value <code>(char *)0</code>, memory is allocated using <code>malloc()</code> to contain the translated character-coded labels. The translated <i>label</i> is copied into allocated or pre-allocated memory.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td><code>LONG_WORDS</code></td><td>Translate using long names of words defined in <i>label</i>.</td></tr> <tr> <td><code>SHORT_WORDS</code></td><td>Translate using short names of words defined in <i>label</i>. If no short name is defined in the <code>label_encodings</code> file for a word, the long name is used.</td></tr> </table>	<code>LONG_WORDS</code>	Translate using long names of words defined in <i>label</i> .	<code>SHORT_WORDS</code>	Translate using short names of words defined in <i>label</i> . If no short name is defined in the <code>label_encodings</code> file for a word, the long name is used.
<code>LONG_WORDS</code>	Translate using long names of words defined in <i>label</i> .				
<code>SHORT_WORDS</code>	Translate using short names of words defined in <i>label</i> . If no short name is defined in the <code>label_encodings</code> file for a word, the long name is used.				

LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the label_encodings file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name 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.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

INFORMATION LABEL [SENSITIVITY LABEL]

The applicable *flags* are LONG_WORDS or SHORT_WORDS , and VIEW_EXTERNAL or VIEW_INTERNAL . A *flags* value 0 is equivalent to (LONG_WORDS).

bsltos() 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).

biltos() translates a binary information label into a string. The applicable *flags* are LONG_CLASSIFICATION or SHORT_CLASSIFICATION , LONG_WORDS or SHORT_WORDS , ALL_ENTRIES or ACCESS_RELATED , VIEW_EXTERNAL or VIEW_INTERNAL , and NO_CLASSIFICATION . A *flags* value 0 is equivalent to (LONG_CLASSIFICATION | LONG_WORDS | ALL_ENTRIES).

bclearaos() 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 may not be the same as the translation of a sensitivity label. These functions

ATTRIBUTES

use different `label_encodings` file tables that may contain different words and constraints.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These routines return:

- 1 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 the `label_encodings` file is inaccessible.
- 0 If memory cannot be allocated for the return string, or the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the `NULL` string (`*string[0]='\00'`).
- >0 If successful, the length of the character-coded label including the `NULL` terminator.

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.

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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`bcltobanner(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltype(3)`, `blvalid(3)`, `btohex(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`, `label_encodings(4)`

Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual****NOTES**

`free(3C)` , `malloc(3C)` , `attributes(5)`

If memory is allocated by these routines, the caller must free the memory with `free()` when the memory is no longer in use.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW` .

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]` ; however, the IL component is fixed at `ADMIN_LOW` .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * label); void bclhigh(bclabel_t * label); void bsllow(bslabel_t * label); void bsllhigh(bslabel_t * label); void bclearlow(bclear_t * clearance); void bclearhigh(bclear_t * clearance); void bclundef(bclabel_t * label); void bsllundef(bslabel_t * label); void bclearundef(bclabel_t * label);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef() , bsllundef() , and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blvalid, bslvalid, bilvalid, bclearvalid – Check validity of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h> int bslvalid(const bslabel_t * label); int bclearvalid(const bclear_t * clearance);</pre>						
DESCRIPTION	<p>The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current process' sensitivity label.</p> <p>These functions check the validity of binary labels.</p> <p>bslvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system.</p> <p>bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <p>-1 If the label_encodings file is inaccessible.</p> <p>0 If the binary label is not valid for this system or is not dominated by the process' sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges,</p> <p>1 If the binary label is valid for this system.</p>						
FILES	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
SEE ALSO	<p>bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) , blmanifest(3) , blminmax(3) , blportion(3) , bltocolour(3) , bltos(3) , bltype(3) , btohex(3) , hextob(3) , labelinfo(3) , labelvers(3) , sbltos(3) , stobl(3) , label_encodings(4)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Binary sensitivity labels are *valid* if they are contained in the `SYSTEM_ACCREDITATION_RANGE` as checked by `blinset(3)`. `bslvalid()` is a synonym for calling `blinset()` with the containing set of `SYSTEM_ACCREDITATION_RANGE` and is included for completeness.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> . bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> .</p> <p>bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	bind – Bind a name to a socket																									
SYNOPSIS	cc [flags...] file ... -lsocket -lnsl [library...] #include <sys/types.h> #include <sys/socket.h> int bind(int s, const struct sockaddr *name, socklen_t *namelen);																									
DESCRIPTION	bind() assigns a name to an unnamed socket. When a socket is created with socket(3N), it exists in a name space (address family) but has no name assigned. bind() requests that the name pointed to by name be assigned to the socket.																									
RETURN VALUES	bind() returns: 0 On success. -1 On failure, and sets errno to indicate the error.																									
ERRORS	The bind() call will fail if: <table><tr><td>EACCES</td><td>The requested address is protected and the current user has inadequate permission to access it.</td></tr><tr><td>EADDRINUSE</td><td>The specified address is already in use.</td></tr><tr><td>EADDRNOTAVAIL</td><td>The specified address is not available on the local machine.</td></tr><tr><td>EBADF</td><td>s is not a valid descriptor.</td></tr><tr><td>EINVAL</td><td>namelen is not the size of a valid address for the specified address family.</td></tr><tr><td>EINVAL</td><td>The socket is already bound to an address.</td></tr><tr><td>ENOSR</td><td>There were insufficient STREAMS resources for the operation to complete.</td></tr><tr><td>ENOTSOCK</td><td>s is a descriptor for a file, not a socket.</td></tr></table> <p>The following errors are specific to binding names in the UNIX domain:</p> <table><tr><td>EACCES</td><td>Search permission is denied for a component of the path prefix of the pathname in name.</td></tr><tr><td>EIO</td><td>An I/O error occurred while making the directory entry or allocating the inode.</td></tr><tr><td>EISDIR</td><td>A null pathname was specified.</td></tr><tr><td>ELOOP</td><td>Too many symbolic links were encountered in translating the pathname in name.</td></tr></table>		EACCES	The requested address is protected and the current user has inadequate permission to access it.	EADDRINUSE	The specified address is already in use.	EADDRNOTAVAIL	The specified address is not available on the local machine.	EBADF	s is not a valid descriptor.	EINVAL	namelen is not the size of a valid address for the specified address family.	EINVAL	The socket is already bound to an address.	ENOSR	There were insufficient STREAMS resources for the operation to complete.	ENOTSOCK	s is a descriptor for a file, not a socket.	EACCES	Search permission is denied for a component of the path prefix of the pathname in name.	EIO	An I/O error occurred while making the directory entry or allocating the inode.	EISDIR	A null pathname was specified.	ELOOP	Too many symbolic links were encountered in translating the pathname in name.
EACCES	The requested address is protected and the current user has inadequate permission to access it.																									
EADDRINUSE	The specified address is already in use.																									
EADDRNOTAVAIL	The specified address is not available on the local machine.																									
EBADF	s is not a valid descriptor.																									
EINVAL	namelen is not the size of a valid address for the specified address family.																									
EINVAL	The socket is already bound to an address.																									
ENOSR	There were insufficient STREAMS resources for the operation to complete.																									
ENOTSOCK	s is a descriptor for a file, not a socket.																									
EACCES	Search permission is denied for a component of the path prefix of the pathname in name.																									
EIO	An I/O error occurred while making the directory entry or allocating the inode.																									
EISDIR	A null pathname was specified.																									
ELOOP	Too many symbolic links were encountered in translating the pathname in name.																									

ENOENT A component of the path prefix of the pathname in *name* does not exist.

ENOTDIR A component of the path prefix of the pathname in *name* is not a directory.

EROFS The inode would reside on a read-only file system.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the calling process possesses the `PRIV_NET_MAC_READ` privilege, the socket is bound to a multilevel port (MLP); otherwise, the socket is bound to a single-level port (SLP).

SEE ALSO

Trusted Solaris 7
Reference Manual

`unlink(2)`

SunOS 5.7 Reference
Manual

`socket(3N)`, `attributes(5)`, `socket(5)`

NOTES

Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using `unlink(2)`).

The rules used in name binding vary between communication domains.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> .</p> <p>bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	bcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> .</p> <p>bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]</pre> <pre>#include <tsol/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> .</p> <p>bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bildominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i> .</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i> .</p> <p>bildominates() compares information label <i>label1</i> for dominance over information label <i>label2</i> .</p> <p>bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i> .</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blinset – Check binary label for inclusion in set						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]</pre> <pre>#include <tsol/label.h> typedef label_set_identifier { int type; char *name; } set_id;</pre> <pre>int blinset(const bslabel_t *label, const set_id *id);</pre>						
DESCRIPTION	<p>The calling process must have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges to perform tests on labels that dominate the current processes' sensitivity label.</p> <p><i>label</i> is examined to determine if it is an element of the label set <i>id</i>. The <i>set_id type</i> field contains a manifest constant defining the type of set to be examined. The <i>set_id name</i> field contains the name of the particular set of type <i>type</i>. The following types and names are defined:</p> <p><code>SYSTEM_ACCREDITATION_RANGE</code> The system's accreditation range as defined in the <code>label_encodings</code> file. The <i>name</i> field is ignored and need not be specified.</p> <p><code>USER_ACCREDITATION_RANGE</code> The user accreditation range as defined in the <code>label_encodings</code> file. The <i>name</i> field is ignored and need not be specified.</p> <p>Other <i>type</i> and <i>name</i> values are reserved for future implementation.</p>						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p><code>blinset()</code> returns:</p> <p>1 If the label is contained in the specified set.</p> <p>0 If the label is not in the specified set, is not a valid sensitivity label, or is not dominated by the process sensitivity label and the process does not have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges.</p> <p>-1 If the specified set is inaccessible.</p>						
FILES	<code>/etc/security/tsol/label_encodings</code>						

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

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blmanifest(3),
blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3),
labelinfo(3), labelvers(3), sbltos(3), stobl(3), label_encodings(4)

Trusted Solaris Developer's Guide and the *Trusted Solaris administrator's
document set*

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

The ADMIN_HIGH and ADMIN_LOW labels are accepted even if the remainder of the system's accreditation range is inaccessible.

BUGS

The only sets available are the System Accreditation Range and User Accreditation Range.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * label); void bclhigh(bclabel_t * label); void bsllow(bslabel_t * label); void bsllhigh(bslabel_t * label); void bclearlow(bclear_t * clearance); void bclearhigh(bclear_t * clearance); void bclundef(bclabel_t * label); void bsllundef(bslabel_t * label); void bclearundef(bclabel_t * label);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef() , bsllundef() , and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME blminmax, blmaximum, blminimum – Bound of two binary levels

SYNOPSIS `cc [flag...] file... -ltsol [library...]`

```
#include <tsol/label.h>
void blmaximum(blevel_t * maximum_label, const blevel_t * bounding_label);
void blminimum(blevel_t * minimum_label, const blevel_t * bounding_label);
```

DESCRIPTION blmaximum() replaces the contents of binary level *maximum_label* with the least upper bound of binary levels *maximum_label* and *bounding_label*. The least upper bound is the greater of the classifications and all of the compartments of the two binary levels. This is the least binary level that dominates both the original binary levels.

blminimum() replaces the contents of binary level *minimum_label* with the greatest lower bound of binary levels *minimum_label* and *bounding_label*. The greatest lower bound is the lower of the classifications and only the compartments contained in both binary levels. This is the greatest binary level that is dominated by both the original binary levels.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

SEE ALSO

Trusted Solaris 7
Reference Manual

bcltobanner(3), blcompare(3), blinset(3), blmanifest(3),
blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3),
labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

attributes(5)

NAME	blminmax, blmaximum, blminimum – Bound of two binary levels						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>void blmaximum(blevel_t * <i>maximum_label</i>, const blevel_t * <i>bounding_label</i>);</pre> <pre>void blminimum(blevel_t * <i>minimum_label</i>, const blevel_t * <i>bounding_label</i>);</pre>						
DESCRIPTION	<p>blmaximum() replaces the contents of binary level <i>maximum_label</i> with the least upper bound of binary levels <i>maximum_label</i> and <i>bounding_label</i> . The least upper bound is the greater of the classifications and all of the compartments of the two binary levels. This is the least binary level that dominates both the original binary levels.</p> <p>blminimum() replaces the contents of binary level <i>minimum_label</i> with the greatest lower bound of binary levels <i>minimum_label</i> and <i>bounding_label</i> . The greatest lower bound is the lower of the classifications and only the compartments contained in both binary levels. This is the greatest binary level that is dominated by both the original binary levels.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO Trusted Solaris 7 Reference Manual SunOS 5.7 Reference Manual	<p>bcltobanner(3) , blcompare(3) , blinset(3) , blmanifest(3) , blportion(3) , bltos(3) , bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) , labelvers(3) , sbltos(3) , stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p> <p>attributes(5)</p>						

NAME blminmax, blmaximum, blminimum – Bound of two binary levels

SYNOPSIS `cc [flag...] file... -ltsol [library...]`

```
#include <tsol/label.h>
void blmaximum(blevel_t * maximum_label, const blevel_t * bounding_label);
void blminimum(blevel_t * minimum_label, const blevel_t * bounding_label);
```

DESCRIPTION blmaximum() replaces the contents of binary level *maximum_label* with the least upper bound of binary levels *maximum_label* and *bounding_label*. The least upper bound is the greater of the classifications and all of the compartments of the two binary levels. This is the least binary level that dominates both the original binary levels.

blminimum() replaces the contents of binary level *minimum_label* with the greatest lower bound of binary levels *minimum_label* and *bounding_label*. The greatest lower bound is the lower of the classifications and only the compartments contained in both binary levels. This is the greatest binary level that is dominated by both the original binary levels.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

SEE ALSO

Trusted Solaris 7
Reference Manual

bcltobanner(3), blcompare(3), blinset(3), blmanifest(3),
blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hex tob(3),
labelinfo(3), labelvers(3), sb ltos(3), stobl(3)

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

attributes(5)

NAME	blportion, bcltosl, bcltoil, biltollev, getcsl, getcil, setcsl, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> bslabel_t * bcltosl(bslabel_t * label); void getcsl(bslabel_t * destination_label, const bslabel_t * source_label); void setcsl(bslabel_t * destination_label, const bslabel_t * source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() and bcltoil() provide a pointer to the sensitivity label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcsl() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcsl() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blegal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), bltocolour(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blcompare, blequal, bilequal, bimequal, bldominates, blstrictdom, bldominates, bimdominates, blinrange – Compare binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...] #include <tso1/label.h> typedef binary_level_range { blevel_t lower_bound; blevel_t upper_bound; } brange_t; int blequal(const blevel_t * label1, const blevel_t * label2); int bimequal(const blevel_t * label1, const blevel_t * label2); int bldominates(const blevel_t * label1, const blevel_t * label2); int blstrictdom(const blevel_t * label1, const blevel_t * label2); int bimdominates(const blevel_t * label1, const blevel_t * label2); int blinrange(const blevel_t * label, const brange_t * range);</pre>						
DESCRIPTION	<p>These functions compare binary labels for meeting a particular condition.</p> <p>blequal() compares two levels for equality.</p> <p>bldominates() compares level <i>label1</i> for dominance over level <i>label2</i>.</p> <p>blstrictdom() compares level <i>label1</i> for strict dominance over level <i>label2</i>.</p> <p>. bldominates() compares information label <i>label1</i> for dominance over information label <i>label2</i>. bimdominates() compares information label markings sets <i>label1</i> for dominance over information label markings sets <i>label2</i>.</p> <p>blinrange() compares level <i>label</i> for dominance over <i>range</i> → <i>lower_bound</i> and <i>range</i> → <i>upper_bound</i> for dominance over level <i>label</i>.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if their respective conditions are met, otherwise zero is returned.						
EXAMPLES	<p>EXAMPLE 1 Compare two binary Labels</p> <p>The following example shows how to compare all parts of two binary labels:</p> <pre>blequal(bcltosl(&cmw_label1), bcltosl(&cmw_label2)) && bilequal(bcltoil(&cmw_label1), bcltoil(&cmw_label2))</pre>						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	bltcolor, bltcolor_r – Get character-coded color name of label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>char * bltcolor(const blevel_t * label);</pre> <pre>char * bltcolor_r(const blevel_t * label, const int size, char * color_name);</pre>						
DESCRIPTION	<p>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 processes' sensitivity label.</p> <p>bltcolor() and bltcolor_r() get the character-coded color name associated with the binary label <i>label</i> .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions described in NOTES</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe with exceptions described in NOTES
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe with exceptions described in NOTES						
RETURN VALUES	<p>bltcolor() returns a pointer to a statically allocated string that contains the character-coded color name specified for the <i>label</i> or returns (char *)0 if, for any reason, no character-coded color name is available for this binary label.</p> <p>bltcolor_r() returns a pointer to the <i>color_name</i> string which contains the character-coded color name specified for the <i>label</i> or returns (char *)0 if, for any reason, no character-coded color name is available for this binary label. <i>color_name</i> must provide for a string of at least <i>size</i> characters.</p>						
FILES	<pre>/etc/security/tsol/ label_encodings</pre> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
SEE ALSO							
Trusted Solaris 7 Reference Manual	<pre>bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) ,</pre> <pre>blmanifest(3) , blminmax(3) , blportion(3) , bltos(3) , bltype(3) ,</pre> <pre>blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) , labelvers(3) ,</pre> <pre>sbltos(3) , stobl(3) , label_encodings(4)</pre> <p><i>Trusted Solaris Developer's Guide</i> , <i>Trusted Solaris User's Guide</i> , and <i>Trusted Solaris administrator's document set</i></p>						
SunOS 5.7 Reference Manual	<pre>attributes(5)</pre>						

NOTES

The function `bltcolor()` 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.

For multithreaded applications the function `bltcolor_r()` should be used.

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.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	bltocolor, bltocolor_r – Get character-coded color name of label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>char * bltocolor(const blevel_t * label);</pre> <pre>char * bltocolor_r(const blevel_t * label, const int size, char * color_name);</pre>						
DESCRIPTION	<p>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 processes' sensitivity label.</p> <p>bltocolor() and bltocolor_r() get the character-coded color name associated with the binary label <i>label</i>.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe with exceptions described in NOTES</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe with exceptions described in NOTES
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe with exceptions described in NOTES						
RETURN VALUES	<p>bltocolor() returns a pointer to a statically allocated string that contains the character-coded color name specified for the <i>label</i> or returns (char *)0 if, for any reason, no character-coded color name is available for this binary label.</p> <p>bltocolor_r() returns a pointer to the <i>color_name</i> string which contains the character-coded color name specified for the <i>label</i> or returns (char *)0 if, for any reason, no character-coded color name is available for this binary label. <i>color_name</i> must provide for a string of at least <i>size</i> characters.</p>						
FILES	<p>/etc/security/tsol/ label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
SEE ALSO							
Trusted Solaris 7 Reference Manual	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3), label_encodings(4)</p> <p><i>Trusted Solaris Developer's Guide</i>, <i>Trusted Solaris User's Guide</i>, and <i>Trusted Solaris administrator's document set</i></p>						
SunOS 5.7 Reference Manual	attributes(5)						

NOTES

The function `bltcolor()` 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.

For multithreaded applications the function `bltcolor_r()` should be used.

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.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	bltos, bcltos, bsltos, biltos, bcleartos – Translate binary labels to character coded labels				
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> int bltos(const blevel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bsltos(const bslabel_t * label, char ** string, const int str_len, const int flags); int bcleartos(const bclear_t * label, char ** string, const int str_len, const int flags);</pre>				
DESCRIPTION	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <p><i>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</i></p> <p>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.</p> <p><i>string</i> may point to either a pointer to pre-allocated memory, or the value (char *) 0 . If it points to a pointer to pre-allocated memory, then <i>str_len</i> indicates the size of that memory. If it points to the value (char *) 0 , memory is allocated using malloc() to contain the translated character-coded labels. The translated <i>label</i> is copied into allocated or pre-allocated memory.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i> .</td></tr> <tr> <td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr> </table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.
LONG_WORDS	Translate using long names of words defined in <i>label</i> .				
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.				

LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the label_encodings file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name 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.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

INFORMATION LABEL [SENSITIVITY LABEL]

The applicable *flags* are LONG_WORDS or SHORT_WORDS , and VIEW_EXTERNAL or VIEW_INTERNAL . A *flags* value 0 is equivalent to (LONG_WORDS).

bsltos() 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).

biltos() translates a binary information label into a string. The applicable *flags* are LONG_CLASSIFICATION or SHORT_CLASSIFICATION , LONG_WORDS or SHORT_WORDS , ALL_ENTRIES or ACCESS_RELATED , VIEW_EXTERNAL or VIEW_INTERNAL , and NO_CLASSIFICATION . A *flags* value 0 is equivalent to (LONG_CLASSIFICATION | LONG_WORDS | ALL_ENTRIES).

bcleartos() 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 may not be the same as the translation of a sensitivity label. These functions

use different `label_encodings` file tables that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These routines return:

- 1 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 the `label_encodings` file is inaccessible.
- 0 If memory cannot be allocated for the return string, or the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the NULL string (`*string[0]='\00'`).
- >0 If successful, the length of the character-coded label including the NULL terminator.

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.

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.

SEE ALSO

Trusted Solaris 7
Reference Manual

`bcltobanner(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltype(3)`, `blvalid(3)`, `btohex(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`, `label_encodings(4)`

Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual****NOTES**

`free(3C)`, `malloc(3C)`, `attributes(5)`

If memory is allocated by these routines, the caller must free the memory with `free()` when the memory is no longer in use.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	bltype, setbltype – Compare and set the type of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>int bltype(const void * <i>label</i>, const unsigned char <i>type</i>);</pre> <pre>void setbltype(void * <i>label</i>, const unsigned char <i>type</i>);</pre>						
DESCRIPTION	<p>These functions compare and set the type of binary labels.</p> <p><code>bltype()</code> examines <i>label</i> to determine if it is of the specified type <i>type</i>.</p> <p><code>setbltype()</code> sets the type of <i>label</i> to the specified type <i>type</i>.</p> <p><i>type</i> may be one of:</p> <p><code>SUN_SL_ID</code> <i>label</i> is a defined binary sensitivity label.</p> <p><code>SUN_SL_UN</code> <i>label</i> is an undefined binary sensitivity label.</p> <p><code>SUN_CLR_ID</code> <i>label</i> is a defined binary clearance.</p> <p><code>SUN_CLR_UN</code> <i>label</i> is an undefined binary clearance.</p> <p><code>SUN_CMW_ID</code> <i>label</i> is a binary CMW label whose sensitivity label and information label portions may or may not be defined. (<code>bltype()</code> only.)</p>						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<code>bltype()</code> returns non-zero if <i>label</i> is of type <i>type</i> , otherwise zero is returned.						
SEE ALSO	<p>Trusted Solaris 7 Reference Manual</p> <p><code>bcltobanner(3)</code>, <code>bilconjoin(3)</code>, <code>blcompare(3)</code>, <code>blinset(3)</code>, <code>blmanifest(3)</code>, <code>blminmax(3)</code>, <code>blportion(3)</code>, <code>bltcolor(3)</code>, <code>bltos(3)</code>, <code>blvalid(3)</code>, <code>btohex(3)</code>, <code>hextob(3)</code>, <code>labelinfo(3)</code>, <code>labelvers(3)</code>, <code>sbltos(3)</code>, <code>stobl(3)</code></p> <p><i>Trusted Solaris Developer's Guide</i></p>						

SunOS 5.7 Reference
Manual

WARNINGS

attributes(5)

- `bltype(&cmw_label, SUN_CMW_ID)` checks the existence of a binary CMW label structure and not the portions of the structure that contain defined labels.
- When attempting to determine the type of a label, rather than to verify that a specific label type is present, check `SUN_CMW_ID` first.
- `setbltype()` makes no checks on the structure it is setting or the type value.

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	blvalid, bslvalid, bilvalid, bclearvalid – Check validity of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h> int bslvalid(const bslabel_t * label); int bclearvalid(const bclear_t * clearance);</pre>						
DESCRIPTION	<p>The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current process' sensitivity label.</p> <p>These functions check the validity of binary labels.</p> <p>bslvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system.</p> <p>bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <p>-1 If the label_encodings file is inaccessible.</p> <p>0 If the binary label is not valid for this system or is not dominated by the process' sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges,</p> <p>1 If the binary label is valid for this system.</p>						
FILES	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
SEE ALSO	<p>bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) , blmanifest(3) , blminmax(3) , blportion(3) , bltocolour(3) , bltos(3) , bltype(3) , btohex(3) , hextob(3) , labelinfo(3) , labelvers(3) , sbltos(3) , stobl(3) , label_encodings(4)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Binary sensitivity labels are *valid* if they are contained in the `SYSTEM_ACCREDITATION_RANGE` as checked by `blinset(3)`. `bslvalid()` is a synonym for calling `blinset()` with the containing set of `SYSTEM_ACCREDITATION_RANGE` and is included for completeness.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bsllabel_t * <i>label</i>); void bsllhigh(bsllabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bsllabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bsllundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bslabel_t * <i>label</i>); void bsllhigh(bslabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bsllundef(bslabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bsllundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	btohex, bcltoh, bsltoh, biltoh, bcleartoh, bcltoh_r, bsltoh_r, biltoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char * bcltoh(const bclabel_t * label); char * bsltoh(const bslabel_t * label); char * bcleartoh(const bclear_t * clearance); char * bcltoh_r(const bclabel_t * label, char * hex); char * bsltoh_r(const bslabel_t * label, char * hex); char * bcleartoh_r(const bclear_t * clearance, char * hex); char * h_alloc(const unsigned char type); void h_free(char * hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), biltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

`h_free()` 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

`atohexlabel(1M)`, `hextoalabel(1M)`, `bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltocolor(3)`, `bltos(3)`, `bltype(3)`, `blvalid(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

The functions `bcltoh()`, `bsltoh()`, `biltoh()`, and `bcleartoh()` 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.

For multithreaded applications, the functions `bcltoh_r()`, `bsltoh_r()`, `biltoh_r()`, and `bcleartoh_r()` should be used.

Information labels (ILs) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any ILs on communications and files from systems running earlier releases as ADMIN_LOW.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 has the following characteristics:

- ILs do not display in window labels; SLs (Sensitivity Labels) display alone within brackets.
- ILs do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	btohex, bcltoh, bsltoh, bilttoh, bcleartoh, bcltoh_r, bsltoh_r, bilttoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char *bcltoh(const bclabel_t *label); char *bsltoh(const bslabel_t *label); char *bcleartoh(const bclear_t *clearance); char *bcltoh_r(const bclabel_t *label, char *hex); char *bsltoh_r(const bslabel_t *label, char *hex); char *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), bilttoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

`h_free()` 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

`atohexlabel(1M)`, `hextoalabel(1M)`, `bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltos(3)`, `bltype(3)`, `blvalid(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

The functions `bcltoh()`, `bsltoh()`, `biltoh()`, and `bcleartoh()` 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.

For multithreaded applications, the functions `bcltoh_r()`, `bsltoh_r()`, `biltoh_r()`, and `bcleartoh_r()` should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW.

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL]; however, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	bltos, bcltos, bsltos, biltos, bcleartos – Translate binary labels to character coded labels				
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> int bltos(const blevel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bcltos(const bclabel_t * label, char ** string, const int str_len, const int flags); int bsltos(const bslabel_t * label, char ** string, const int str_len, const int flags); int bcleartos(const bclear_t * label, char ** string, const int str_len, const int flags);</pre>				
DESCRIPTION	<p>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' sensitivity label.</p> <p>These routines translate binary labels into strings controlled by the value of the <i>flags</i> parameter.</p> <p>The generic form of an output character-coded label is:</p> <p><i>CLASSIFICATION WORD1 WORD2 WORD3/WORD4 SUFFIX PREFIX WORD5/WORD6</i></p> <p>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.</p> <p><i>string</i> may point to either a pointer to pre-allocated memory, or the value (char *) 0 . If it points to a pointer to pre-allocated memory, then <i>str_len</i> indicates the size of that memory. If it points to the value (char *) 0 , memory is allocated using malloc() to contain the translated character-coded labels. The translated <i>label</i> is copied into allocated or pre-allocated memory.</p> <p><i>flags</i> is 0 (zero), or the logical sum of the following:</p> <table> <tr> <td>LONG_WORDS</td><td>Translate using long names of words defined in <i>label</i> .</td></tr> <tr> <td>SHORT_WORDS</td><td>Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.</td></tr> </table>	LONG_WORDS	Translate using long names of words defined in <i>label</i> .	SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.
LONG_WORDS	Translate using long names of words defined in <i>label</i> .				
SHORT_WORDS	Translate using short names of words defined in <i>label</i> . If no short name is defined in the label_encodings file for a word, the long name is used.				

LONG_CLASSIFICATION	Translate using long name of classification defined in <i>label</i> .
SHORT_CLASSIFICATION	Translate using short name of classification defined in <i>label</i> .
ACCESS_RELATED	Translate only <i>access-related</i> entries defined in information label <i>label</i> .
VIEW_EXTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the lowest and highest labels defined in the label_encodings file.
VIEW_INTERNAL	Translate ADMIN_LOW and ADMIN_HIGH labels to the admin low name 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.
NO_CLASSIFICATION	Do not translate classification defined in <i>label</i> .

bcltos() translates a binary CMW label into a string of the form:

INFORMATION LABEL [SENSITIVITY LABEL]

The applicable *flags* are LONG_WORDS or SHORT_WORDS , and VIEW_EXTERNAL or VIEW_INTERNAL . A *flags* value 0 is equivalent to (LONG_WORDS).

bsltos() 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).

biltos() translates a binary information label into a string. The applicable *flags* are LONG_CLASSIFICATION or SHORT_CLASSIFICATION , LONG_WORDS or SHORT_WORDS , ALL_ENTRIES or ACCESS_RELATED , VIEW_EXTERNAL or VIEW_INTERNAL , and NO_CLASSIFICATION . A *flags* value 0 is equivalent to (LONG_CLASSIFICATION | LONG_WORDS | ALL_ENTRIES).

bclearaos() 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 may not be the same as the translation of a sensitivity label. These functions

use different `label_encodings` file tables that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These routines return:

- 1 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 the `label_encodings` file is inaccessible.
- 0 If memory cannot be allocated for the return string, or the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the NULL string (`*string[0]='\00'`).
- >0 If successful, the length of the character-coded label including the NULL terminator.

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.

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.

SEE ALSO

Trusted Solaris 7
Reference Manual

`bcltobanner(3)`, `blcompare(3)`, `blinset(3)`, `blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltocolour(3)`, `bltype(3)`, `blvalid(3)`, `btohex(3)`, `hextob(3)`, `labelinfo(3)`, `labelvers(3)`, `sbltos(3)`, `stobl(3)`, `label_encodings(4)`

Trusted Solaris Developer's Guide, Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual
NOTES**`free(3C) , malloc(3C) , attributes(5)`

If memory is allocated by these routines, the caller must free the memory with `free()` when the memory is no longer in use.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW` .

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]` ; however, the IL component is fixed at `ADMIN_LOW` .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	blmanifest, bcllow, bclhigh, bsllow, bsllhigh, billow, bilhigh, bclearlow, bclearhigh, bclundef, bslundef, bilundef, bclearundef – Create manifest binary labels						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> void bcllow(bclabel_t * <i>label</i>); void bclhigh(bclabel_t * <i>label</i>); void bsllow(bslabel_t * <i>label</i>); void bsllhigh(bslabel_t * <i>label</i>); void bclearlow(bclear_t * <i>clearance</i>); void bclearhigh(bclear_t * <i>clearance</i>); void bclundef(bclabel_t * <i>label</i>); void bslundef(bslabel_t * <i>label</i>); void bclearundef(bclear_t * <i>label</i>);</pre>						
DESCRIPTION	<p>These functions initialize <i>binary label</i> structures to manifest values.</p> <p>bcllow() and bclhigh() initialize the binary CMW label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH CMW labels, respectively.</p> <p>bsllow() and bsllhigh() initialize the binary sensitivity label structure <i>label</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH sensitivity labels, respectively.</p> <p>bclearlow() and bclearhigh() initialize the binary clearance structure <i>clearance</i> to the manifest constant values for the ADMIN_LOW and ADMIN_HIGH clearances, respectively.</p> <p>bclundef(), bslundef(), and bilundef() initialize the binary CMW , sensitivity, and information label structure <i>label</i> to the manifest constant value for an undefined CMW , sensitivity, and information label, respectively.</p> <p>bclearundef() initializes the binary clearance <i>clearance</i> to the manifest constant value for an undefined clearance.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blminmax(3), blportion(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blvalid, bslvalid, bilvalid, bclearvalid – Check validity of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h> int bslvalid(const bslabel_t * label); int bclearvalid(const bclear_t * clearance);</pre>						
DESCRIPTION	<p>The calling process must have PRIV_SYS_TRANS_LABEL in its set of effective privileges to inquire about labels that dominate the current process' sensitivity label.</p> <p>These functions check the validity of binary labels.</p> <p>bslvalid() examines <i>label</i> to determine if it is a valid sensitivity label for this system.</p> <p>bclearvalid() examines <i>clearance</i> to determine if it is a valid clearance for this system.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <p>-1 If the label_encodings file is inaccessible.</p> <p>0 If the binary label is not valid for this system or is not dominated by the process' sensitivity label and the process does not have PRIV_SYS_TRANS_LABEL in its set of effective privileges,</p> <p>1 If the binary label is valid for this system.</p>						
FILES	<p>/etc/security/tsol/label_encodings</p> <p>The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
SEE ALSO	<p>bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) , blmanifest(3) , blminmax(3) , blportion(3) , bltocolour(3) , bltos(3) , bltype(3) , btohex(3) , hextob(3) , labelinfo(3) , labelvers(3) , sbltos(3) , stobl(3) , label_encodings(4)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Binary sensitivity labels are *valid* if they are contained in the `SYSTEM_ACCREDITATION_RANGE` as checked by `blinset(3)`. `bslvalid()` is a synonym for calling `blinset()` with the containing set of `SYSTEM_ACCREDITATION_RANGE` and is included for completeness.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW`, and cannot be set on any objects.

NAME	btohex, bcltoh, bsltoh, biltoh, bcleartoh, bcltoh_r, bsltoh_r, biltoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char * bcltoh(const bclabel_t * label); char * bsltoh(const bslabel_t * label); char * bcleartoh(const bclear_t * clearance); char * bcltoh_r(const bclabel_t * label, char * hex); char * bsltoh_r(const bslabel_t * label, char * hex); char * bcleartoh_r(const bclear_t * clearance, char * hex); char * h_alloc(const unsigned char type); void h_free(char * hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), biltoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

h_free() 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

The functions bcltoh(), bsltoh(), biltoh(), and bcleartoh() 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.

For multithreaded applications, the functions bcltoh_r(), bsltoh_r(), biltoh_r(), and bcleartoh_r() should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – Translate and verify user authorizations
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id); char auth_set_to_str(auth_set_t * authset, char separator); auth_t str_to_auth(char * auth_name); auth_set_t *str_to_auth_set(char * auth_names, char * separators); char *get_auth_text(auth_t auth_id); int chkauth(auth_t auth_id, char * username); void free_auth_set(auth_set_t * authset);</pre>
DESCRIPTION	<p>auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i> . If <i>auth_id</i> is an undefined authorization ID , the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID , TSOL_MAX_AUTH , a NULL is returned.</p> <p>auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C) . Authorization names are separated by the character <i>separator</i> . Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list.</p> <p>str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i> . <i>auth_name</i> is the name in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH .</p> <p>str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i> . <i>auth_names</i> are names in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . The string none is translated to an empty authorization list (NULL) . This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization</p>

RETURN VALUES

names appear in the *auth_names* string, the function converts these invalid authorizations into *auth_id*s with the value of 0 .

get_auth_text() returns a pointer to the statically-allocated, null-terminated authorization description text specified by *auth_id* .

chkauth() returns 1 if the user, specified by *username* has the authorization specified by *auth_id* .

free_auth_set() releases the memory returned by *str_to_auth_set*() .

auth_to_str() returns a pointer to the translated authorization name string. It returns NULL on failure.

str_to_auth() returns the numeric authorization id. It returns 0 on failure.

auth_set_to_str() returns a pointer to the translated authorization names string. It returns NULL on failure.

str_to_auth_set() returns NULL on success.

get_auth_text() return a string on success and NULL upon failure.

chkauth() returns 1 on success and 0 on failure.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

NOTES

This interface is uncommitted, which means it may change between minor releases of the Trusted Solaris environment. To use these routines, the program must be loaded with the Trusted Solaris library *libtsolddb* .

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auth_desc(4) , *auth_name*(4)

attributes(5)

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

`void clnt_perrno(const enum clnt_stat stat);`

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

`void clnt_perror(const CLIENT * clnt, const char * s);`

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

`char *clnt_sperrno(const enum clnt_stat stat);`

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

`char *clnt_sperror(const CLIENT * clnt, const char * s);`

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

`enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char * nettype);`

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles</p>
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t      get the RPC program's version
                    number associated with the
                    client handle
CLSET_VERS  rpcvers_t set the RPC program's version
                    number associated with the
                    client handle. This assumes
                    that the RPC server for this
                    new version is still listening
                    at the address of the previous
                    version.
CLGET_XID  uint32_t get the XID of the previous
                    remote procedure call
CLSET_XID  uint32_t set the XID of the next
                    remote procedure call
CLGET_PROG rpcprog_t get program number
CLSET_PROG rpcprog_t set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated `AUTH` structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_screateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildes, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildes* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle.  This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated `AUTH` structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_screateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle.  This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
rcvsvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by *clnt_call()* (see *clnt_call()* in *rpc_clnt_calls(3N)*). The retry time out and the total time out periods can be changed using *clnt_control()*. The user may set the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see *svc_raw_create()* in *rpc_svc_create(3N)*). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. *clnt_raw_create()* should be called after *svc_raw_create()*.

```
char *clnt_spcreateerror(const char * s);
```

Like *clnt_pcreateerror()*, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t rcvsvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC_UNKNOWNADDR error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle. This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated `AUTH` structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_spcreateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t      get the RPC program's version
                    number associated with the
                    client handle
CLSET_VERS  rpcvers_t      set the RPC program's version
                    number associated with the
                    client handle. This assumes
                    that the RPC server for this
                    new version is still listening
                    at the address of the previous
                    version.
CLGET_XID   uint32_t       get the XID of the previous
                    remote procedure call
CLSET_XID   uint32_t       set the XID of the next
                    remote procedure call
CLGET_PROG  rpcprog_t      get program number
CLSET_PROG  rpcprog_t      set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
rcvsvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_spcreateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildes, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t rcvsvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildes* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle.  This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t      get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t      set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t     get program number
CLSET_PROG  rpcprog_t     set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

`clnt_control()` returns TRUE on success and FALSE on failure.

CLIENT *`clnt_create`(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

`clnt_create()` tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using `clnt_control()`. This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the rpcbind service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`).

CLIENT *`clnt_create_timed`(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated `AUTH` structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_spcreateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles</p>
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle.  This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_spcreateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

void clnt_perrno(const enum clnt_stat *stat*);

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

void clnt_perror(const CLIENT * *clnt*, const char * *s*);

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

char *clnt_sperrno(const enum clnt_stat *stat*);

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

char *clnt_sperror(const CLIENT * *clnt*, const char * *s*);

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

enum clnt_stat rpc_broadcast(const rpcprog_t *prognum*, const rpcvers_t *versnum*, const rpcproc_t *procnum*, const xdrproc_t *inproc*, const caddr_t *in*, const xdrproc_t *outproc*, caddr_t *out*, const resultproc_t *eachresult*, const char * *nettype*);

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to `"netpath"`. Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

void clnt_perrno(const enum clnt_stat *stat*);

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

void clnt_perror(const CLIENT * *clnt*, const char * *s*);

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

char *clnt_sperrno(const enum clnt_stat *stat*);

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

char *clnt_sperror(const CLIENT * *clnt*, const char * *s*);

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

enum clnt_stat rpc_broadcast(const rpcprog_t *prognum*, const rpcvers_t *versnum*, const rpcproc_t *procnum*, const xdrproc_t *inproc*, const caddr_t *in*, const xdrproc_t *outproc*, caddr_t *out*, const resultproc_t *eachresult*, const char * *nettype*);

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles</p>
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle.  This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns NULL. A default timeout is set and can be modified using `clnt_control()`. This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by *clnt_call()* (see *clnt_call()* in *rpc_clnt_calls(3N)*). The retry time out and the total time out periods can be changed using *clnt_control()*. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see *svc_raw_create()* in *rpc_svc_create(3N)*). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. *clnt_raw_create()* should be called after *svc_raw_create()*.

```
char *clnt_spcreateerror(const char * s);
```

Like *clnt_pcreateerror()*, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC_UNKNOWNADDR error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

```
void clnt_perrno(const enum clnt_stat stat);
```

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
void clnt_perror(const CLIENT * clnt, const char * s);
```

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
char *clnt_sperrno(const enum clnt_stat stat);
```

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

```
char *clnt_sperror(const CLIENT * clnt, const char * s);
```

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum,
const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const
xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char * nettype);
```

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:


```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

`void clnt_perrno(const enum clnt_stat stat);`

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

`void clnt_perror(const CLIENT * clnt, const char * s);`

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

`char *clnt_sperrno(const enum clnt_stat stat);`

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

`char *clnt_sperror(const CLIENT * clnt, const char * s);`

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

`enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char * nettype);`

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	<p>rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles</p>
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle.  This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by *clnt_call()* (see *clnt_call()* in *rpc_clnt_calls(3N)*). The retry time out and the total time out periods can be changed using *clnt_control()*. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see *svc_raw_create()* in *rpc_svc_create(3N)*). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. *clnt_raw_create()* should be called after *svc_raw_create()*.

```
char *clnt_spcreateerror(const char * s);
```

Like *clnt_pcreateerror()*, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC_UNKNOWNADDR error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spcreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t      get the RPC program's version
                    number associated with the
                    client handle
CLSET_VERS  rpcvers_t      set the RPC program's version
                    number associated with the
                    client handle.  This assumes
                    that the RPC server for this
                    new version is still listening
                    at the address of the previous
                    version.
CLGET_XID   uint32_t       get the XID of the previous
                    remote procedure call
CLSET_XID   uint32_t       set the XID of the next
                    remote procedure call
CLGET_PROG  rpcprog_t      get program number
CLSET_PROG  rpcprog_t      set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns NULL. A default timeout is set and can be modified using `clnt_control()`. This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_spcreateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

`void clnt_perrno(const enum clnt_stat stat);`

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

`void clnt_perror(const CLIENT *clnt, const char *s);`

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

`char *clnt_sperrno(const enum clnt_stat stat);`

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

`char *clnt_sperror(const CLIENT *clnt, const char *s);`

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

`enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char *nettype);`

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

`void clnt_perrno(const enum clnt_stat stat);`

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

`void clnt_perror(const CLIENT * clnt, const char * s);`

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

`char *clnt_sperrno(const enum clnt_stat stat);`

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

`char *clnt_sperror(const CLIENT * clnt, const char * s);`

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

`enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char * nettype);`

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:


```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t      get the RPC program's version
                    number associated with the
                    client handle
CLSET_VERS  rpcvers_t      set the RPC program's version
                    number associated with the
                    client handle.  This assumes
                    that the RPC server for this
                    new version is still listening
                    at the address of the previous
                    version.
CLGET_XID   uint32_t       get the XID of the previous
                    remote procedure call
CLSET_XID   uint32_t       set the XID of the next
                    remote procedure call
CLGET_PROG  rpcprog_t      get program number
CLSET_PROG  rpcprog_t      set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated `AUTH` structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
rcvsvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by *clnt_call()* (see *clnt_call()* in *rpc_clnt_calls(3N)*). The retry time out and the total time out periods can be changed using *clnt_control()*. The user may set the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see *svc_raw_create()* in *rpc_svc_create(3N)*). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. *clnt_raw_create()* should be called after *svc_raw_create()*.

```
char *clnt_spcreateerror(const char * s);
```

Like *clnt_pcreateerror()*, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t rcvsvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC_UNKNOWNADDR error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle.  This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated `AUTH` structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_screateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, `RPC_UNKNOWNADDR` error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <pre>bool_t clnt_control(CLIENT * clnt , const uint_t req , char * info) ;</pre> <p>A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t    get the RPC program's version
                  number associated with the
                  client handle
CLSET_VERS  rpcvers_t    set the RPC program's version
                  number associated with the
                  client handle. This assumes
                  that the RPC server for this
                  new version is still listening
                  at the address of the previous
                  version.
CLGET_XID   uint32_t     get the XID of the previous
                  remote procedure call
CLSET_XID   uint32_t     set the XID of the next
                  remote procedure call
CLGET_PROG  rpcprog_t    get program number
CLSET_PROG  rpcprog_t    set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

`clnt_control()` returns TRUE on success and FALSE on failure.

CLIENT *`clnt_create`(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

`clnt_create()` tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using `clnt_control()`. This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the rpcbind service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`).

CLIENT *`clnt_create_timed`(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildes, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_screateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildes, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC_UNKNOWNADDR error is set. *fildes* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spccreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a <code>CLIENT</code> handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code>.</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>CLIENT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code> A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t      get the RPC program's version
                    number associated with the
                    client handle
CLSET_VERS  rpcvers_t      set the RPC program's version
                    number associated with the
                    client handle. This assumes
                    that the RPC server for this
                    new version is still listening
                    at the address of the previous
                    version.
CLGET_XID   uint32_t        get the XID of the previous
                    remote procedure call
CLSET_XID   uint32_t        set the XID of the next
                    remote procedure call
CLGET_PROG  rpcprog_t      get program number
CLSET_PROG  rpcprog_t      set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control() returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

clnt_create() tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using clnt_control(). This routine returns NULL if it fails. The clnt_pcreateerror() routine can be used to print the reason for failure.

Note: clnt_create() returns a valid client handle even if the particular version number supplied to clnt_create() is not registered with the rpcbind service. This mismatch will be discovered by a clnt_call later (see rpc_clnt_calls(3N)).

CLIENT *clnt_create_timed(const char * *host*, const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*, const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

```
CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t
* vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype );
```

Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns `NULL`. A default timeout is set and can be modified using `clnt_control()`. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

```
CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum,
rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *
nettype const struct timeval * timeout);
```

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

```
void clnt_destroy(CLIENT * clnt );
```

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated `AUTH` structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by *clnt_call()* (see *clnt_call()* in *rpc_clnt_calls(3N)*). The retry time out and the total time out periods can be changed using *clnt_control()*. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see *svc_raw_create()* in *rpc_svc_create(3N)*). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. *clnt_raw_create()* should be called after *svc_raw_create()*.

```
char *clnt_spcreateerror(const char * s);
```

Like *clnt_pcreateerror()*, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC_UNKNOWNADDR error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	clock_settime, clock_gettime, clock_getres – High-resolution clock operations
SYNOPSIS	<pre>cc [flags...] file ... -lrt [library...] #include <time.h> int clock_settime(clockid_t clock_id, const struct timespec * tp); int clock_gettime(clockid_t clock_id, struct timespec * tp); int clock_getres(clockid_t clock_id, struct timespec * res);</pre>
DESCRIPTION	<p>The <code>clock_settime()</code> function sets the specified clock, <code>clock_id</code>, to the value specified by <code>tp</code>. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p>The <code>clock_gettime()</code> function returns the current value <code>tp</code> for the specified clock, <code>clock_id</code>.</p> <p>The resolution of any clock can be obtained by calling <code>clock_getres()</code>. Clock resolutions are system-dependent and cannot be set by a process. If the argument <code>res</code> is not <code>NULL</code>, the resolution of the specified clock is stored in the location pointed to by <code>res</code>. If <code>res</code> is <code>NULL</code>, the clock resolution is not returned. If the time argument of <code>clock_settime()</code> is not a multiple of <code>res</code>, then the value is truncated to a multiple of <code>res</code>.</p> <p>A clock may be systemwide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). A <code>clock_id</code> of <code>CLOCK_REALTIME</code> is defined in <code><time.h></code>. This clock represents the realtime clock for the system. For this clock, the values returned by <code>clock_gettime()</code> and specified by <code>clock_settime()</code> represent the amount of time (in seconds and nanoseconds) since the Epoch. Additional clocks may also be supported. The interpretation of time values for these clocks is unspecified.</p>
RETURN VALUES	<p><code>clock_settime()</code> returns:</p> <ul style="list-style-type: none"> 0 On success. -1 On failure, and sets <code>errno</code> to indicate the error.
ERRORS	<p>The <code>clock_settime()</code>, <code>clock_gettime()</code> and <code>clock_getres()</code> functions will fail if:</p> <ul style="list-style-type: none"> <code>EINVAL</code> The <code>clock_id</code> argument does not specify a known clock. <code>ENOSYS</code> The functions <code>clock_settime()</code>, <code>clock_gettime()</code>, and <code>clock_getres()</code> are not supported by this implementation.

The `clock_settime()` function will fail if:

`EINVAL` The *tp* argument to `clock_settime()` is outside the range for the given clock ID ; or the *tp* argument specified a nanosecond value less than zero or greater than or equal to 1000 million.

The `clock_settime()` function may fail if:

`EPERM` The requesting process does not have the appropriate privilege to set the specified clock.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The calling process must have the `PRIV_SYS_CONFIG` privilege in order to set the specified clock.

SEE ALSO SunOS 5.7 Reference Manual

`time(2)`, `ctime(3C)`, `timer_gettime(3C)`, `attributes(5)`, `time(5)`

NAME	clock_settime, clock_gettime, clock_getres – High-resolution clock operations
SYNOPSIS	<pre>cc [flags...] file ... -lrt [library...] #include <time.h> int clock_settime(clockid_t clock_id, const struct timespec * tp); int clock_gettime(clockid_t clock_id, struct timespec * tp); int clock_getres(clockid_t clock_id, struct timespec * res);</pre>
DESCRIPTION	<p>The <code>clock_settime()</code> function sets the specified clock, <code>clock_id</code>, to the value specified by <code>tp</code>. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p>The <code>clock_gettime()</code> function returns the current value <code>tp</code> for the specified clock, <code>clock_id</code>.</p> <p>The resolution of any clock can be obtained by calling <code>clock_getres()</code>. Clock resolutions are system-dependent and cannot be set by a process. If the argument <code>res</code> is not <code>NULL</code>, the resolution of the specified clock is stored in the location pointed to by <code>res</code>. If <code>res</code> is <code>NULL</code>, the clock resolution is not returned. If the time argument of <code>clock_settime()</code> is not a multiple of <code>res</code>, then the value is truncated to a multiple of <code>res</code>.</p> <p>A clock may be systemwide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). A <code>clock_id</code> of <code>CLOCK_REALTIME</code> is defined in <code><time.h></code>. This clock represents the realtime clock for the system. For this clock, the values returned by <code>clock_gettime()</code> and specified by <code>clock_settime()</code> represent the amount of time (in seconds and nanoseconds) since the Epoch. Additional clocks may also be supported. The interpretation of time values for these clocks is unspecified.</p>
RETURN VALUES	<p><code>clock_settime()</code> returns:</p> <ul style="list-style-type: none"> 0 On success. -1 On failure, and sets <code>errno</code> to indicate the error.
ERRORS	<p>The <code>clock_settime()</code>, <code>clock_gettime()</code> and <code>clock_getres()</code> functions will fail if:</p> <ul style="list-style-type: none"> <code>EINVAL</code> The <code>clock_id</code> argument does not specify a known clock. <code>ENOSYS</code> The functions <code>clock_settime()</code>, <code>clock_gettime()</code>, and <code>clock_getres()</code> are not supported by this implementation.

The `clock_settime()` function will fail if:

EINVAL The *tp* argument to `clock_settime()` is outside the range for the given clock ID ; or the *tp* argument specified a nanosecond value less than zero or greater than or equal to 1000 million.

The `clock_settime()` function may fail if:

EPERM The requesting process does not have the appropriate privilege to set the specified clock.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The calling process must have the `PRIV_SYS_CONFIG` privilege in order to set the specified clock.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`time(2)`, `ctime(3C)`, `timer_gettime(3C)`, `attributes(5)`, `time(5)`

NAME	clock_settime, clock_gettime, clock_getres – High-resolution clock operations
SYNOPSIS	<pre>cc [flags...] file ... -lrt [library...] #include <time.h> int clock_settime(clockid_t clock_id, const struct timespec * tp); int clock_gettime(clockid_t clock_id, struct timespec * tp); int clock_getres(clockid_t clock_id, struct timespec * res);</pre>
DESCRIPTION	<p>The <code>clock_settime()</code> function sets the specified clock, <code>clock_id</code>, to the value specified by <code>tp</code>. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to set the specified clock.</p> <p>The <code>clock_gettime()</code> function returns the current value <code>tp</code> for the specified clock, <code>clock_id</code>.</p> <p>The resolution of any clock can be obtained by calling <code>clock_getres()</code>. Clock resolutions are system-dependent and cannot be set by a process. If the argument <code>res</code> is not <code>NULL</code>, the resolution of the specified clock is stored in the location pointed to by <code>res</code>. If <code>res</code> is <code>NULL</code>, the clock resolution is not returned. If the time argument of <code>clock_settime()</code> is not a multiple of <code>res</code>, then the value is truncated to a multiple of <code>res</code>.</p> <p>A clock may be systemwide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). A <code>clock_id</code> of <code>CLOCK_REALTIME</code> is defined in <code><time.h></code>. This clock represents the realtime clock for the system. For this clock, the values returned by <code>clock_gettime()</code> and specified by <code>clock_settime()</code> represent the amount of time (in seconds and nanoseconds) since the Epoch. Additional clocks may also be supported. The interpretation of time values for these clocks is unspecified.</p>
RETURN VALUES	<p><code>clock_settime()</code> returns:</p> <ul style="list-style-type: none"> 0 On success. -1 On failure, and sets <code>errno</code> to indicate the error.
ERRORS	<p>The <code>clock_settime()</code>, <code>clock_gettime()</code> and <code>clock_getres()</code> functions will fail if:</p> <ul style="list-style-type: none"> <code>EINVAL</code> The <code>clock_id</code> argument does not specify a known clock. <code>ENOSYS</code> The functions <code>clock_settime()</code>, <code>clock_gettime()</code>, and <code>clock_getres()</code> are not supported by this implementation.

The `clock_settime()` function will fail if:

`EINVAL` The *tp* argument to `clock_settime()` is outside the range for the given clock ID ; or the *tp* argument specified a nanosecond value less than zero or greater than or equal to 1000 million.

The `clock_settime()` function may fail if:

`EPERM` The requesting process does not have the appropriate privilege to set the specified clock.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>clock_gettime()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The calling process must have the `PRIV_SYS_CONFIG` privilege in order to set the specified clock.

SEE ALSO SunOS 5.7 Reference Manual

`time(2)`, `ctime(3C)`, `timer_gettime(3C)`, `attributes(5)`, `time(5)`

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search`

`res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery`

The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <code><arpa/nameser.h></code>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

if *lastdnptr* is NULL, `dn_comp()` does not update the list of labels. If *dnptrs* is NULL, names are not compressed.

`res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *tll*, and returns a pointer to a linked list *ns_updrec*. It returns NULL upon failure.

`res_update()` takes a linked list of resource records *ns_updrec* as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns <0 upon error.

`res_mkupdate()` creates update packets by running through the elements of the *ns_updrec* link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the *rdata* portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search` `res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery` The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <code><arpa/nameser.h></code>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

- if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.
- res_mkupdrec** `res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *tll*, and returns a pointer to a linked list *ns_updrec*. It returns `NULL` upon failure.
- res_update** `res_update()` takes a linked list of resource records *ns_updrec* as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.
- res_mkupdate** `res_mkupdate()` creates update packets by running through the elements of the *ns_updrec* link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or
- 1 Error in reading a word or number in the *rdata* portion for update packets.
 - 2 The length of the packet passed is insufficient.
 - 3 The zone section is not the first section in the linked list, or the section order has a problem.
 - 4 A number overflow.
 - 5 Unknown operations or no records.
- If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	door_create – Create a door descriptor				
SYNOPSIS	<pre>cc [flags...] file ... -ldoor -lthread [library...]</pre> <pre>#include <door.h> int door_create(void *server_procedure, void *cookie, char *argp, size_t arg_size, door_desc_t *dp, uint_t n_desc, void *cookie, uint_t attributes);</pre>				
DESCRIPTION	<p>The <code>door_create()</code> function creates a door descriptor that describes the procedure specified by the function <code>server_procedure</code>.</p> <p>In the Trusted Solaris environment, <code>door_create()</code> sets the door CMW label and MAC policy based on attributes of the creating process. The CMW label of the door is set equal to the CMW label of the calling process. If the calling process has the <code>sys_system_door</code> privilege, the MAC policy is set to <code>MAC_ANY</code>, otherwise its MAC policy is set to <code>MAC_EQUAL</code>.</p> <p>The data item, <code>cookie</code>, is associated with the door descriptor, and is passed as an argument to the invoked function <code>server_procedure</code> during <code>door_call(3X)</code> invocations. Other arguments passed to <code>server_procedure</code> from an associated <code>door_call()</code> are placed on the stack and include <code>argp</code> and <code>dp</code>. <code>argp</code> points to <code>arg_size</code> bytes of data and <code>dp</code> points to <code>n_desc</code> <code>door_desc_t</code> structures. The <code>attributes</code> flag specifies attributes associated with the newly created door. Valid values for <code>attributes</code> are constructed by ORing in one or more of the following values:</p> <table> <tr> <td>DOOR_UNREF</td><td>Delivers a special invocation on the door when the number of descriptors that refer to this door drops to one. In order to trigger this condition, more than one descriptor must have referred to this door at some time. <code>DOOR_UNREF_DATA</code> designates an unreferenced invocation, as the <code>argp</code> argument passed to <code>server_procedure</code>. In the case of an unreferenced invocation, the values for <code>arg_size</code>, <code>dp</code> and <code>n_desc</code> are 0. Only one unreferenced invocation is delivered on behalf of a door.</td></tr> <tr> <td>DOOR_UNREF_MULTI</td><td>Similar to <code>DOOR_UNREF</code>, except multiple unreferenced invocations can be delivered on the same door if the number of descriptors referring to the door drops to one more than once. Since an additional reference may have been passed by the time an unreferenced invocation arrives, the <code>DOOR_IS_UNREF</code> attribute returned by the <code>door_info(3X)</code> call can be used to determine if the door is still unreferenced.</td></tr> </table>	DOOR_UNREF	Delivers a special invocation on the door when the number of descriptors that refer to this door drops to one. In order to trigger this condition, more than one descriptor must have referred to this door at some time. <code>DOOR_UNREF_DATA</code> designates an unreferenced invocation, as the <code>argp</code> argument passed to <code>server_procedure</code> . In the case of an unreferenced invocation, the values for <code>arg_size</code> , <code>dp</code> and <code>n_desc</code> are 0. Only one unreferenced invocation is delivered on behalf of a door.	DOOR_UNREF_MULTI	Similar to <code>DOOR_UNREF</code> , except multiple unreferenced invocations can be delivered on the same door if the number of descriptors referring to the door drops to one more than once. Since an additional reference may have been passed by the time an unreferenced invocation arrives, the <code>DOOR_IS_UNREF</code> attribute returned by the <code>door_info(3X)</code> call can be used to determine if the door is still unreferenced.
DOOR_UNREF	Delivers a special invocation on the door when the number of descriptors that refer to this door drops to one. In order to trigger this condition, more than one descriptor must have referred to this door at some time. <code>DOOR_UNREF_DATA</code> designates an unreferenced invocation, as the <code>argp</code> argument passed to <code>server_procedure</code> . In the case of an unreferenced invocation, the values for <code>arg_size</code> , <code>dp</code> and <code>n_desc</code> are 0. Only one unreferenced invocation is delivered on behalf of a door.				
DOOR_UNREF_MULTI	Similar to <code>DOOR_UNREF</code> , except multiple unreferenced invocations can be delivered on the same door if the number of descriptors referring to the door drops to one more than once. Since an additional reference may have been passed by the time an unreferenced invocation arrives, the <code>DOOR_IS_UNREF</code> attribute returned by the <code>door_info(3X)</code> call can be used to determine if the door is still unreferenced.				

DOOR_PRIVATE Maintains a separate pool of server threads on behalf of the door. Server threads are associated with a door's private server pool using door_bind(3X).

The descriptor returned from door_create() will be marked as close on exec (FD_CLOEXEC). Information about a door is available for all clients of a door using door_info(3X). Programs concerned with security should not place secure information in door data that is accessible by door_info(). In particular, secure data should not be stored in the data item *cookie*.

By default, additional threads are created as needed to handle concurrent door_call(3X) invocations. See door_server_create(3X) for information on how to change this behavior.

RETURN VALUES

door_create() returns:

>0 On success.

-1 On failure, and sets errno to indicate the error.

ERRORS

The door_create() function will fail if:

EINVAL Invalid attributes are passed.

EMFILE The process has too many open descriptors.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	all
Availability	SUNWcsu
Stability	Evolving
MT-Level	Safe

SUMMARY
OF TRUSTED
SOLARIS
CHANGES

In the Trusted Solaris environment, door_create() sets the door CMW label and MAC policy based on attributes of the creating process. The CMW label of the door is set equal to the CMW label of the calling process. If the calling process has the sys_system_door privilege, the MAC policy is set to MAC_ANY, otherwise its MAC policy is set to MAC_EQUAL.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

door_tcred(3X)

door_bind(3X), door_call(3X), door_info(3X), door_revoke(3X), door_create(3X), attributes(5)

NAME	door_tcred – Return the extended credential information associated with the client of the current door invocation										
SYNOPSIS	<pre>cc [flags...] file ... -ltsol -lthread [library...]</pre>										
DESCRIPTION	<pre>#include <door.h> int door_tcred(door_tcred_t *info);</pre> <p>The door_tcred() function returns the extended credential information associated with the client (if any) of the current door invocation.</p> <p>The tsol_door_cred_t structure is returned by the door_tcred() interface. It is added to the Trusted Solaris environment so that a door server is able to get the Trusted Solaris attributes of the calling client.</p> <pre>/* * Structure used to return info from door_tcred */ typedef struct tsol_door_cred { door_cred_t tdc_cred; /* cred data */ bclabel_t tdc_cmw_label; /* CMW Label */ bclear_t tdc_clearance; /* Clearance */ pattr_t tdc_proc_attr; /* Proc. Attr. Flags */ priv_set_t tdc_effective; /* Effective set */ } tsol_door_cred_t;</pre> <p>The credential information associated with the client refers to the information from the immediate caller, not necessarily from the first thread in a chain of door calls.</p>										
RETURN VALUES	<p>door_tcred() returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets errno to indicate the error.</p>										
ERRORS	<p>The door_tcred() function fails if:</p> <p>EFAULT The address of the <i>info</i> argument is invalid.</p> <p>EINVAL There is no associated door client.</p>										
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Architecture</td><td>all</td></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>Stability</td><td>Unstable</td></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Architecture	all	Availability	SUNWtsu	Stability	Unstable	MT-Level	Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE										
Architecture	all										
Availability	SUNWtsu										
Stability	Unstable										
MT-Level	Safe										

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

door_create(3X)

door_call(3X), door_cred(3X), attributes(5)

It would be more appropriate to use an extensible mechanism rather than the door_tcred() call. This is expected to be part of the general extension mechanism for process attributes, and will be addressed then. The current door_tcred() can be re-implemented in terms of such a general mechanism.

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information		
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int getacdir(char * dir, int len); int getacmin(int * min_val); int getacflg(char * auditstring, int len); int getacna(char * auditstring, int len); void setac(void); void endac(void);</pre>		
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>		
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <p>0 on success.</p> <p>-2 On failure and set <code>errno</code> to indicate the error.</p>		

getacmin() and getacflg() return:

1 On EOF .

getacdir() returns:

-1 on EOF .

2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir() .

These functions return:

-3 If the directory entry format in the audit_control file is incorrect.

getacdir() , getacflg() and getacna() return:

-3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SUMMARY OF TRUSTED SOLARIS CHANGES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

audit_warn(1M) , inetd(1M) , audit_control(4)

SunOS 5.7 Reference
Manual

attributes(5)

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char * name); struct au_class_ent *getauclassnam_r(au_class_ent_t * class_int, const char * name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t * class_int); void setauclass(void); void endauclass(void);</pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i> .</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL .</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent() .</p> <p>endauclass() may be called to indicate that audit_class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t , which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:</p> <pre>char *ac_name; au_class_t ac_class; char *ac_desc;</pre>

RETURN VALUES

getauclassnam() and getauclassnam_r() return a pointer to a struct `au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

getauclassent() and getauclassent_r() return a pointer to a struct `au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

FILES

`/etc/security/audit_class` Maps audit class numbers to audit class names.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except `getauclassent()` and `getauclassnam()`. The two functions, `getauclassent_r()` and `getauclassnam_r()` have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_class(4)`, `audit_event(4)`

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

All information in the MT-unsafe versions are contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL .</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name .</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number .</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r() .</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t      ae_number; char          *ae_name; char
*ae_desc; au_class_t      ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

- `/etc/security/audit_event` Maps audit event numbers to audit event names.
- `/etc/passwd` Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY
OF TRUSTED
SOLARIS
CHANGES

SEE ALSO

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

Trusted Solaris 7 Reference Manual	<code>getauclassent(3)</code> , <code>audit_class(4)</code> , <code>audit_event(4)</code>
SunOS 5.7 Reference Manual	<code>getpwnam(3C)</code> , <code>passwd(4)</code> , <code>attributes(5)</code>
NOTES	All information for the functions <code>getauevent()</code> , <code>getauevnam()</code> , and <code>getauevnum()</code> is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent * getauusernam(const char * name); struct au_user_ent * getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent * getauusernam_r(au_user_ent_t * u, const char * name); struct au_user_ent * getauuserent_r(au_user_ent_t * u);</pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry.</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or <code>NULL</code>.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code><bsm/libbsm.h></code> with the following members:</p> <pre>char *au_name;au_mask_t au_always;au_mask_t au_never;</pre>
RETURN VALUES	The <code>getauusernam()</code> function returns a pointer to an <code>au_user_ent</code> structure if it successfully locates the requested entry; otherwise it returns <code>NULL</code> .

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****ATTRIBUTES**

The `getauuserent()` function returns a pointer to an `au_user_ent` structure if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

FILES

<code>/etc/security/audit_user</code>	Stores per-user audit event mask.
<code>/etc/passwd</code>	Stores user-id to username mappings.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_user(4)`

SunOS 5.7 Reference
Manual

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

NOTES

All information for the `getauuserent()` and `getauusername()` functions is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

The `getauusername()` and `getauuserent()` functions are not MT-safe. The `getauusername_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profent_t * getprofentbyname(char * name, int src); profent_t * getprofent(int src); void setprofent(int stayopen, int src); void endprofent(int src); void free_profent(profent_t * profent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofentbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofent() , getprofent() , and endprofent() are used to enumerate profile entries from the database. setprofent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofent() . A call to getprofentbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofent() .</p> <p>Successive calls to getprofent() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofent() may be called to indicate that the caller expects to do no further profile entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile entry retrieval functions after calling endprofent() .</p> <p>The functions getprofentbyname() and getprofent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profent() should be used to free the pointers returned by either getprofentbyname() or getprofent() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p> <p>The parameter <i>src</i> may be set to any of TSOL_DB_SRC_FILES , TSOL_DB_SRC_NISPLUS , or TSOL_DB_SRC_SWITCH , which are defined</p>

in `<tsol/tsol.h>`. For most applications the `src` parameter should be set to `TSOL_DB_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_DB_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_DB_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofent()`, the threads will enumerate disjoint subsets of the `tsolprof` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `profent_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profent_t {
    char *name;      /* name of profile */
    char *desc;      /* description */
    char *auths;     /* comma separated list of authorization numbers */
    profact_t *actions; /* linked list of actions */
    profcmd_t *cmds; /* linked list of commands */
} profent_t;
```

The function `getprofentbyname()` returns a pointer to a struct `profent_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofent()` returns a pointer to a struct `profent_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

ERRORS

The functions `getprofentbyname()` and `getprofent()` return `NULL` on a failure.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)` ,
Notes On Multithread Applications , for information about the use of the
`_REENTRANT` flag.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

`Intro(3)`

`attributes(5)`

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profstr_t * getprofstrbyname(char * name, int src); profstr_t * getprofstr(int src); int putprofstr(profstr_t * res, int src); int setprofstr(int stayopen, int src); int endprofstr(int src); void free_profstr(profstr_t * profstr);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the <code>tsolprof</code> NIS+ database or the <code>/etc/security/tsol/tsoluser</code> file.</p> <p><code>getprofstrbyname()</code> searches for information for a profile with the specified profile name given by the parameter <i>name</i>.</p> <p>The functions <code>setprofstr()</code>, <code>getprofstr()</code>, and <code>endprofstr()</code> are used to enumerate profile entries from the database. <code>setprofstr()</code> sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to <code>getprofstr()</code>. A call to <code>getprofstrbyname()</code> leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to <code>endprofstr()</code>.</p> <p>Successive calls to <code>getprofstr()</code> return either successive entries or return <code>NULL</code>, indicating the end of the enumeration.</p> <p><code>endprofstr()</code> may be called to indicate that the caller expects to do no further profile string retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile string retrieval functions after calling <code>endprofstr()</code>.</p> <p>The functions <code>getprofstrbyname()</code> and <code>getprofstr()</code> are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function <code>free_profstr()</code> should be used to free the pointers returned by either <code>getprofstrbyname()</code> or <code>getprofstr()</code>.</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p>

The parameter *src* may be set to any of `TSOL_SRC_FILES`, `TSOL_SRC_NISPLUS`, or `TSOL_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`. For most applications the *src* parameter should be set to `TSOL_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

The function `putprofstr()` replaces an existing profile entry or adds a new entry if the profile name does not already exist. Currently the *src* parameter is treated as `TSOL_SRC_NISPLUS`, forcing all information to be written to the NIS+ table. Use of files or of `nsswitch.conf`(4) may be supported in future releases.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofstr()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofstr()`, the threads enumerate disjoint subsets of the `tsolprof`(4) database.

User entries are represented by the struct `profstr_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profstr_t {
    char  name;      /* name of profile */
    char  desc;      /* description */
    char  auths;     /* comma separated list of authorization numbers */
    char  actions;   /* semicolon separated action descriptions */
    char  cmds;      /* semicolon separated command descriptions */
} profstr_t;
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The function `getprofstrbyname()` returns a pointer to a `profstr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofstr()` returns a pointer to a `profstr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The function `putprofstr()` returns 0 on success.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

Intro(3)

attributes(5)

Programs that use the interfaces described here cannot be linked statically since the implementation of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)` , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get Trusted Solaris user security attributes
SYNOPSIS	<pre>cc [flag...] file... -ltsolddb -ltsol -lnsl -lcmd [library...] #include <tsol/user.h> userent_t * getuserentbyname(char * user, int src); userent_t * getuserentbyuid(uid_t uid, int src); userent_t * getuserent(int src); void setuserent(int stayopen, int src); void enduserent(int src); void free_userent(userent_t * userent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user attributes from the tsoluser NIS+ database.</p> <p>getuserentbyname() searches for information for a user with the specified user name <i>user</i>. getuserentbyuid() searches for information for a user with the specified user ID <i>uid</i>.</p> <p>The functions setuserent(), getuserent(), and enduserent() are used to list user entries from the database. setuserent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris user entries. This function should be called before the first call to getuserent(). A call to getuserbyname() or getuserentbyuid() leaves the list position in an indeterminate state. If the <i>stayopen</i> flag is not zero, the system may keep allocated resources such as open file descriptors until a subsequent call to enduserent().</p> <p>Successive calls to getuserent() return either successive entries or NULL, which indicates the end of the list.</p> <p>enduserent() may be called when the caller expects to do no further user-entry-retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more user-entry-retrieval functions after calling enduserent().</p> <p>The functions getuserentbyname(), getuserentbyuid(), and getuserent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_userent() is used to release memory allocated by these two functions. The parameter <i>user</i>, a pointer to the user name, must be a null-terminated character string. The parameter <i>uid</i> must be a uid_t.</p>

The parameter `src` may be set to `TSOL_DB_SRC_FILES`, `TSOL_DB_SRC_NISPLUS`, or `TSOL_DB_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`.

For listing in multithreaded applications, the position within the list is a process-wide-property shared by all threads. `setuserent()` may be used in a multithreaded application but resets the list position for all threads. If multiple threads interleave calls to `getuserent()`, the threads will list disjoint subsets of the `tsoluser` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `userent_t` structure defined in `<tsol/user.h>`:

```
typedef struct userent_s {
    char *name;           /* user associated with this entry */
    char *lock;           /* is account locked? */
    char *badlogins;      /* how many failed login attempts so far */
    char *generation;     /* method of password generation */
    char *profiles;       /* user profiles used */
    char *roles;          /* roles assumable */
    char *idletime;       /* minutes a workstation may remain idle */
    char *idlecmd;        /* what to do at when idletime reached */
    char *labelview;      /* can user see ADMIN_HI and ADMIN_LOW labels */
    char *labeltrans;     /* process security attributes for label translation */
    char *labelmin;       /* allowed low login labels */
    char *labelmax;       /* allowed high login labels */
    char *usertype;       /* normal, admin-role, or non-admin-role */
    char *res1;           /* reserved for future use */
    char *res2;           /* reserved for future use */
    char *res3;           /* reserved for future use */
} userent_t;
```

If it successfully locates the requested entry, `getuserentbyname()` returns a pointer to a `userent_t`. If unsuccessful, `getuserentbyname()` returns `NULL`.

If it successfully lists an entry, `getuserent()` returns a pointer to a struct `userent_t`. If unsuccessful, `getuserent()` returns `NULL`, indicating the end of the list.

Upon success, `setuserent()` and `enduserent()` return 0.

The functions `getuserentbyname()`, `getuserentbyuid()`, and `getuserent()` return `NULL` on failure.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`Intro(3)`

**SunOS 5.7 Reference
Manual**

`attributes(5)`

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically because the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file);</pre>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutent() The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>

pututline()	<p>The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututline()</code> will have searched for the proper entry using one of the these functions. If so, <code>pututline()</code> will not search. If <code>pututline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmp</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>		
setutent()	The <code>setutent()</code> function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.		
endutent()	The <code>endutent()</code> function closes the currently open file.		
utmpname()	The <code>utmpname()</code> function allows the user to change the name of the file examined, from <code>/var/adm/utmp</code> to any other file. It is most often expected that this other file will be <code>/var/adm/wtmp</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpname()</code> function does not open the file but closes the old file if it is currently open and saves the new file name.		
RETURN VALUES	A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, <code>utmpname()</code> returns 0. Otherwise, it returns 1.		
USAGE	<p>These functions use buffered standard I/O for input, but <code>pututline()</code> uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the <code>utmp</code> and <code>wtmp</code> files.</p> <p>Applications should not access the <code>utmp</code> or <code>utmpx</code> database files directly, but should use the functions described on the <code>getutxent(3C)</code> manual page to interact with these files. Using these extended API s will ensure that the <code>utmp</code> and <code>utmpx</code> databases are maintained consistently.</p>		
FILES	<table> <tr> <td><code>/var/adm/utmp</code></td><td>User access and accounting information (old format)</td></tr> </table>	<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmp</code>	User access and accounting information (old format)		

<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

`pututline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututline()` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` functions, if the user has just modified those contents and passed the pointer back to `pututline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmp *utmp, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfile, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

	information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)								
<code>getutmpx()</code>	The <code>getutmpx()</code> function copies the information stored in the members of the <code>utmp</code> structure to the corresponding members of the <code>utmpx</code> structure. (See <code>getutent(3C)</code> for <code>utmp</code> structure)								
<code>updwtmp()</code>	The <code>updwtmp()</code> function checks the existence of <code>wfile</code> and its parallel file, whose name is obtained by appending an “f3x” to <code>wfile</code> . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. <code>utmp</code> is written to <code>wfile</code> and the corresponding <code>utmpx</code> structure is written to the parallel file.								
<code>updwtmpx()</code>	The <code>updwtmpx()</code> function checks the existence of <code>wfilex</code> and its parallel file, whose name is obtained by truncating the final “f3x” from <code>wfilex</code> . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. <code>utmpx</code> is written to <code>wfilex</code> , and the corresponding <code>utmp</code> structure is written to the parallel file.								
RETURN VALUES	A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.								
USAGE	<p>These functions use buffered standard I/O for input, but <code>pututxline()</code> uses an unbuffered write to avoid race conditions between processes trying to modify the <code>utmpx</code> and <code>wtmpx</code> files.</p> <p>Applications should not access the <code>utmp</code> or <code>utmpx</code> database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the <code>utmp</code> and <code>utmpx</code> databases are maintained consistently.</p>								
FILES	<table> <tr> <td><code>/var/adm/utmp</code></td><td>User access and accounting information (old format)</td></tr> <tr> <td><code>/var/adm/utmpx</code></td><td>User access and accounting information (new format)</td></tr> <tr> <td><code>/var/adm/wtmp</code></td><td>History of user access and accounting information (old format)</td></tr> <tr> <td><code>/var/adm/wtmpx</code></td><td>History of user access and accounting information (new format)</td></tr> </table>	<code>/var/adm/utmp</code>	User access and accounting information (old format)	<code>/var/adm/utmpx</code>	User access and accounting information (new format)	<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)	<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)
<code>/var/adm/utmp</code>	User access and accounting information (old format)								
<code>/var/adm/utmpx</code>	User access and accounting information (new format)								
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)								
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)								
ATTRIBUTES	See <code>attributes(5)</code> for descriptions of the following attributes:								
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	Unsafe								

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – Translate and verify user authorizations
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id); char auth_set_to_str(auth_set_t * authset, char separator); auth_t str_to_auth(char * auth_name); auth_set_t *str_to_auth_set(char * auth_names, char * separators); char *get_auth_text(auth_t auth_id); int chkauth(auth_t auth_id, char * username); void free_auth_set(auth_set_t * authset);</pre>
DESCRIPTION	<p>auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i>. If <i>auth_id</i> is an undefined authorization ID, the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID, TSOL_MAX_AUTH, a NULL is returned.</p> <p>auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C). Authorization names are separated by the character <i>separator</i>. Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list.</p> <p>str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i>. <i>auth_name</i> is the name in the Names field of auth_name(4); this function does not parse the names in the Manifest Constant field of auth_name(4). Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH.</p> <p>str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i>. <i>auth_names</i> are names in the Names field of auth_name(4); this function does not parse the names in the Manifest Constant field of auth_name(4). The string none is translated to an empty authorization list (NULL). This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization</p>

names appear in the *auth_names* string, the function converts these invalid authorizations into *auth_id*s with the value of 0 .

get_auth_text() returns a pointer to the statically-allocated, null-terminated authorization description text specified by *auth_id* .

chkauth() returns 1 if the user, specified by *username* has the authorization specified by *auth_id* .

free_auth_set() releases the memory returned by *str_to_auth_set*() .

auth_to_str() returns a pointer to the translated authorization name string. It returns NULL on failure.

str_to_auth() returns the numeric authorization id. It returns 0 on failure.

auth_set_to_str() returns a pointer to the translated authorization names string. It returns NULL on failure.

str_to_auth_set() returns NULL on success.

get_auth_text() return a string on success and NULL upon failure.

chkauth() returns 1 on success and 0 on failure.

RETURN VALUES

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

NOTES

This interface is uncommitted, which means it may change between minor releases of the Trusted Solaris environment. To use these routines, the program must be loaded with the Trusted Solaris library *libtsolddb* .

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auth_desc(4) , *auth_name*(4)

attributes(5)

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profent_t * getprofentbyname(char * name, int src); profent_t * getprofent(int src); void setprofent(int stayopen, int src); void endprofent(int src); void free_profent(profent_t * profent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofentbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofent() , getprofent() , and endprofent() are used to enumerate profile entries from the database. setprofent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofent() . A call to getprofentbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofent() .</p> <p>Successive calls to getprofent() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofent() may be called to indicate that the caller expects to do no further profile entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile entry retrieval functions after calling endprofent() .</p> <p>The functions getprofentbyname() and getprofent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profent() should be used to free the pointers returned by either getprofentbyname() or getprofent() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p> <p>The parameter <i>src</i> may be set to any of TSOL_DB_SRC_FILES , TSOL_DB_SRC_NISPLUS , or TSOL_DB_SRC_SWITCH , which are defined</p>

in `<tsol/tsol.h>`. For most applications the `src` parameter should be set to `TSOL_DB_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_DB_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_DB_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofent()`, the threads will enumerate disjoint subsets of the `tsolprof` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `profent_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profent_t {
    char *name;      /* name of profile */
    char *desc;      /* description */
    char *auths;     /* comma separated list of authorization numbers */
    profact_t *actions; /* linked list of actions */
    profcmd_t *cmds; /* linked list of commands */
} profent_t;
```

The function `getprofentbyname()` returns a pointer to a struct `profent_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofent()` returns a pointer to a struct `profent_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

ERRORS

The functions `getprofentbyname()` and `getprofent()` return `NULL` on a failure.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)` ,
Notes On Multithread Applications , for information about the use of the
`_REENTRANT` flag.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

`Intro(3)`

`attributes(5)`

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profstr_t * getprofstrbyname(char * name, int src); profstr_t * getprofstr(int src); int putprofstr(profstr_t * res, int src); int setprofstr(int stayopen, int src); int endprofstr(int src); void free_profstr(profstr_t * profstr);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the <code>tsolprof</code> NIS+ database or the <code>/etc/security/tsol/tsoluser</code> file.</p> <p><code>getprofstrbyname()</code> searches for information for a profile with the specified profile name given by the parameter <i>name</i>.</p> <p>The functions <code>setprofstr()</code>, <code>getprofstr()</code>, and <code>endprofstr()</code> are used to enumerate profile entries from the database. <code>setprofstr()</code> sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to <code>getprofstr()</code>. A call to <code>getprofstrbyname()</code> leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to <code>endprofstr()</code>.</p> <p>Successive calls to <code>getprofstr()</code> return either successive entries or return <code>NULL</code>, indicating the end of the enumeration.</p> <p><code>endprofstr()</code> may be called to indicate that the caller expects to do no further profile string retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile string retrieval functions after calling <code>endprofstr()</code>.</p> <p>The functions <code>getprofstrbyname()</code> and <code>getprofstr()</code> are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function <code>free_profstr()</code> should be used to free the pointers returned by either <code>getprofstrbyname()</code> or <code>getprofstr()</code>.</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p>

The parameter *src* may be set to any of `TSOL_SRC_FILES`, `TSOL_SRC_NISPLUS`, or `TSOL_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`. For most applications the *src* parameter should be set to `TSOL_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

The function `putprofstr()` replaces an existing profile entry or adds a new entry if the profile name does not already exist. Currently the *src* parameter is treated as `TSOL_SRC_NISPLUS`, forcing all information to be written to the NIS+ table. Use of files or of `nsswitch.conf(4)` may be supported in future releases.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofstr()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofstr()`, the threads enumerate disjoint subsets of the `tsolprof(4)` database.

User entries are represented by the struct `profstr_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profstr_t {
    char  name;      /* name of profile */
    char  desc;      /* description */
    char  auths;     /* comma separated list of authorization numbers */
    char  actions;   /* semicolon separated action descriptions */
    char  cmds;      /* semicolon separated command descriptions */
} profstr_t;
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The function `getprofstrbyname()` returns a pointer to a `profstr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofstr()` returns a pointer to a `profstr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The function `putprofstr()` returns 0 on success.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

Intro(3)

attributes(5)

Programs that use the interfaces described here cannot be linked statically since the implementation of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see Intro(3) , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get Trusted Solaris user security attributes
SYNOPSIS	<pre>cc [flag...] file... -ltsolddb -ltsol -lnsl -lcmd [library...] #include <tsol/user.h> userent_t * getuserentbyname(char * user, int src); userent_t * getuserentbyuid(uid_t uid, int src); userent_t * getuserent(int src); void setuserent(int stayopen, int src); void enduserent(int src); void free_userent(userent_t * userent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user attributes from the <code>tsoluser</code> NIS+ database.</p> <p><code>getuserentbyname()</code> searches for information for a user with the specified user name <i>user</i>. <code>getuserentbyuid()</code> searches for information for a user with the specified user ID <i>uid</i>.</p> <p>The functions <code>setuserent()</code>, <code>getuserent()</code>, and <code>enduserent()</code> are used to list user entries from the database. <code>setuserent()</code> sets (or resets) the enumeration to the beginning of the set of Trusted Solaris user entries. This function should be called before the first call to <code>getuserent()</code>. A call to <code>getuserbyname()</code> or <code>getuserentbyuid()</code> leaves the list position in an indeterminate state. If the <i>stayopen</i> flag is not zero, the system may keep allocated resources such as open file descriptors until a subsequent call to <code>enduserent()</code>.</p> <p>Successive calls to <code>getuserent()</code> return either successive entries or <code>NULL</code>, which indicates the end of the list.</p> <p><code>enduserent()</code> may be called when the caller expects to do no further user-entry-retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more user-entry-retrieval functions after calling <code>enduserent()</code>.</p> <p>The functions <code>getuserentbyname()</code>, <code>getuserentbyuid()</code>, and <code>getuserent()</code> are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function <code>free_userent()</code> is used to release memory allocated by these two functions. The parameter <i>user</i>, a pointer to the user name, must be a null-terminated character string. The parameter <i>uid</i> must be a <code>uid_t</code>.</p>

The parameter `src` may be set to `TSOL_DB_SRC_FILES`, `TSOL_DB_SRC_NISPLUS`, or `TSOL_DB_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`.

For listing in multithreaded applications, the position within the list is a process-wide-property shared by all threads. `setuserent()` may be used in a multithreaded application but resets the list position for all threads. If multiple threads interleave calls to `getuserent()`, the threads will list disjoint subsets of the `tsoluser` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `userent_t` structure defined in `<tsol/user.h>`:

```
typedef struct userent_s {
    char *name;           /* user associated with this entry */
    char *lock;           /* is account locked? */
    char *badlogins;      /* how many failed login attempts so far */
    char *generation;     /* method of password generation */
    char *profiles;       /* user profiles used */
    char *roles;          /* roles assumable */
    char *idletime;       /* minutes a workstation may remain idle */
    char *idlecmd;        /* what to do at when idletime reached */
    char *labelview;      /* can user see ADMIN_HI and ADMIN_LOW labels */
    char *labeltrans;     /* process security attributes for label translation */
    char *labelmin;       /* allowed low login labels */
    char *labelmax;       /* allowed high login labels */
    char *usertype;       /* normal, admin-role, or non-admin-role */
    char *res1;           /* reserved for future use */
    char *res2;           /* reserved for future use */
    char *res3;           /* reserved for future use */
} userent_t;
```

If it successfully locates the requested entry, `getuserentbyname()` returns a pointer to a `userent_t`. If unsuccessful, `getuserentbyname()` returns `NULL`.

If it successfully lists an entry, `getuserent()` returns a pointer to a struct `userent_t`. If unsuccessful, `getuserent()` returns `NULL`, indicating the end of the list.

Upon success, `setuserent()` and `enduserent()` return 0.

The functions `getuserentbyname()`, `getuserentbyuid()`, and `getuserent()` return `NULL` on failure.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`Intro(3)`

**SunOS 5.7 Reference
Manual**

`attributes(5)`

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically because the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	ftw, nftw – Walk a file tree																		
SYNOPSIS	<pre>#include <ftw.h> int ftw(const char * path, int (* fn) (const char *, const struct stat *, int), int depth); int nftw(const char * path, int (* fn) (const char *, const struct stat *, int, struct FTW*), int depth, int flags);</pre>																		
DESCRIPTION	<p>The <code>ftw()</code> function recursively descends the directory hierarchy rooted in <i>path</i>. For each object in the hierarchy, <code>ftw()</code> calls the user-defined function <i>fn</i>, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a <code>stat</code> structure (see <code>stat(2)</code>) containing information about the object, and an integer. Possible values of the integer, defined in the <code><ftw.h></code> header, are:</p> <table> <tr> <td>FTW_F</td><td>The object is a file.</td></tr> <tr> <td>FTW_D</td><td>The object is a directory.</td></tr> <tr> <td>FTW_DNR</td><td>The object is a directory that cannot be read. Descendants of the directory will not be processed.</td></tr> <tr> <td>FTW_NS</td><td>The <code>stat()</code> function failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The <code>stat</code> buffer passed to <i>fn</i> is undefined.</td></tr> </table> <p>The <code>ftw()</code> function visits a directory before visiting any of its descendants.</p> <p>The tree traversal continues until the tree is exhausted, an invocation of <i>fn</i> returns a non-zero value, or some error is detected within <code>ftw()</code> (such as an I/O error). If the tree is exhausted, <code>ftw()</code> returns 0. If <i>fn</i> returns a non-zero value, <code>ftw()</code> stops its tree traversal and returns whatever value was returned by <i>fn</i>.</p> <p><code>nftw(3C)</code> recursively descends the directory hierarchy rooted in <i>path</i>. The <i>flags</i> argument is used to control the tree walk and holds one of these values:</p> <table> <tr> <td>FTW_PHYS</td><td>Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> will follow links but will not walk down any path that crosses itself.</td></tr> <tr> <td>FTW_MOUNT</td><td>The walk will not cross a mount point.</td></tr> <tr> <td>FTW_DEPTH</td><td>All subdirectories will be visited before the directory itself.</td></tr> <tr> <td>FTW_CHDIR</td><td>The walk will change to each directory before reading it.</td></tr> <tr> <td>FTW_TSOL_MLD</td><td>In all multilevel directories (MLD s) encountered as <code>nftw()</code> walks the tree, the walk will visit single-level directories (SLD s) that are dominated by the sensitivity label of the process if the process is run without privilege. If the effective privilege set of the process includes the</td></tr> </table>	FTW_F	The object is a file.	FTW_D	The object is a directory.	FTW_DNR	The object is a directory that cannot be read. Descendants of the directory will not be processed.	FTW_NS	The <code>stat()</code> function failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The <code>stat</code> buffer passed to <i>fn</i> is undefined.	FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> will follow links but will not walk down any path that crosses itself.	FTW_MOUNT	The walk will not cross a mount point.	FTW_DEPTH	All subdirectories will be visited before the directory itself.	FTW_CHDIR	The walk will change to each directory before reading it.	FTW_TSOL_MLD	In all multilevel directories (MLD s) encountered as <code>nftw()</code> walks the tree, the walk will visit single-level directories (SLD s) that are dominated by the sensitivity label of the process if the process is run without privilege. If the effective privilege set of the process includes the
FTW_F	The object is a file.																		
FTW_D	The object is a directory.																		
FTW_DNR	The object is a directory that cannot be read. Descendants of the directory will not be processed.																		
FTW_NS	The <code>stat()</code> function failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The <code>stat</code> buffer passed to <i>fn</i> is undefined.																		
FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> will follow links but will not walk down any path that crosses itself.																		
FTW_MOUNT	The walk will not cross a mount point.																		
FTW_DEPTH	All subdirectories will be visited before the directory itself.																		
FTW_CHDIR	The walk will change to each directory before reading it.																		
FTW_TSOL_MLD	In all multilevel directories (MLD s) encountered as <code>nftw()</code> walks the tree, the walk will visit single-level directories (SLD s) that are dominated by the sensitivity label of the process if the process is run without privilege. If the effective privilege set of the process includes the																		

`PRIV_FILE_MAC_READ` and `PRIV_FILE_MAC_SEARCH` privileges, the walk visits all SLD s in each MLD . The file system enforces all underlying DAC policies and privilege interpretations.

If the `FTW_TSOL_MLD` flag is *not* specified and *path* points to an adorned MLD , the walk traverses only the SLD s of this MLD . All other MLD s encountered while walking down the tree are automatically translated to the SLD at the sensitivity label of the process even if the process is run with all privileges.

If the `FTW_TSOL_MLD` flag is *not* specified and *path* points to an unadorned MLD , when the walk down the tree encounters this and all other MLD s, then the function automatically translates to the SLD at the sensitivity label of the process.

If the `FTW_TSOL_MLD` flag is *not* specified and *path* does not point to an MLD , when the walk down the tree encounters any MLD s, then the function automatically translates to the SLD at the sensitivity label of the process even if the process is run with all privileges.

The `nftw()` function calls *fn* with four arguments at each file and directory. The first argument is the pathname of the object, the second is a pointer to the `stat` structure [see `stat(2)`] containing information about the object, the third is an integer giving additional information, and the fourth is a `struct FTW` that contains the following members:

```
int    base;
int    level;
```

The `base` member is the offset into the pathname of the base name of the object. The `level` member indicates the depth relative to the rest of the walk, where the root level is zero.

The values of the third argument are as follows:

<code>FTW_F</code>	The object is a file.
<code>FTW_D</code>	The object is a directory.
<code>FTW_DP</code>	The object is a directory and subdirectories have been visited.
<code>FTW_SL</code>	The object is a symbolic link.
<code>FTW_SLN</code>	The object is a symbolic link that points to a non-existent file.

FTW_DNR	The object is a directory that cannot be read.]The user-defined function <i>fn</i> will not be called for any of its descendants.
FTW_NS	The <code>stat()</code> function failed on the object because of lack of appropriate permission. The <code>stat</code> buffer passed to <i>fn</i> is undefined. The <code>stat()</code> function failed for a reason other than lack of appropriate permission. <code>EACCES</code> is considered an error and <code>nftw()</code> will return <code>-1</code> .

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error (such as an I/O error) is detected within `nftw()` . If the tree is exhausted, `nftw()` returns zero. If *fn* returns a nonzero value, `nftw()` stops its tree traversal and returns whatever value *fn* returned.

Both `ftw()` and `nftw()` use one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were `1` . It must not be greater than the number of file descriptors currently available for use. The `ftw()` function will run faster if *depth* is at least as large as the number of levels in the tree. When `ftw()` and `nftw()` return, they close any file descriptors they have opened; they do not close any file descriptors that may have been opened by *fn* .

RETURN VALUES

`ftw()` and `nftw()` return:

0	On success.
-1	If either function detects an error other than <code>EACCES</code> , and sets <code>errno</code> to indicate the error.

USAGE

Because `ftw()` is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

The `ftw()` function uses `malloc(3C)` to allocate dynamic storage during its operation. If `ftw()` is forcibly terminated, such as by `longjmp(3C)` being executed by *fn* or an interrupt routine, `ftw()` will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a non-zero value at its next invocation.

The `ftw()` and `nftw()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)` .

The `ftw()` function is safe in multithreaded applications. The `nftw()` function is safe in multithreaded applications when the `FTW_CHDIR` flag is not set.

There are two versions of `nftw()` . The Solaris version, which does not traverse multilevel directories (`MLD` s), is located in `libc` ; the Trusted Solaris version,

SUMMARY OF TRUSTED SOLARIS CHANGES

ATTRIBUTES

which traverses MLD s, is located in `libtsol`. To use the Trusted Solaris version of `nftw()`, make sure that the application uses the version of `nftw` located in `libtsol`.

The `libc` versions of `ftw()` and `nftw()` are unchanged. The Trusted Solaris version of `nftw()`, which has the additional flag `FTW_TSOL_MLD`, is available in `libtsol`. You must be careful of the library sequence when linking.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe with exceptions.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`stat(2)`

`longjmp(3C)`, `malloc(3C)`, `attributes(5)`, `lf64(5)`

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information		
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int getacdir(char * dir, int len); int getacmin(int * min_val); int getacflg(char * auditstring, int len); int getacna(char * auditstring, int len); void setac(void); void endac(void);</pre>		
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>		
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <p>0 on success.</p> <p>-2 On failure and set <code>errno</code> to indicate the error.</p>		

getacmin() and getacflg() return:

1 On EOF .

getacdir() returns:

-1 on EOF .

2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir() .

These functions return:

-3 If the directory entry format in the audit_control file is incorrect.

getacdir() , getacflg() and getacna() return:

-3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SUMMARY OF TRUSTED SOLARIS CHANGES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

audit_warn(1M) , inetd(1M) , audit_control(4)

SunOS 5.7 Reference
Manual

attributes(5)

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information		
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int getacdir(char * dir, int len); int getacmin(int * min_val); int getacflg(char * auditstring, int len); int getacna(char * auditstring, int len); void setac(void); void endac(void);</pre>		
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>		
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <ul style="list-style-type: none"> 0 on success. -2 On failure and set <code>errno</code> to indicate the error. 		

getacmin() and getacflg() return:

1 On EOF .

getacdir() returns:

-1 on EOF .

2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir() .

These functions return:

-3 If the directory entry format in the audit_control file is incorrect.

getacdir() , getacflg() and getacna() return:

-3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SUMMARY OF TRUSTED SOLARIS CHANGES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

audit_warn(1M) , inetd(1M) , audit_control(4)

SunOS 5.7 Reference
Manual

attributes(5)

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information		
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int getacdir(char * dir, int len); int getacmin(int * min_val); int getacflg(char * auditstring, int len); int getacna(char * auditstring, int len); void setac(void); void endac(void);</pre>		
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>		
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <p>0 on success.</p> <p>-2 On failure and set <code>errno</code> to indicate the error.</p>		

getacmin() and getacflg() return:

1 On EOF .

getacdir() returns:

-1 on EOF .

2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir() .

These functions return:

-3 If the directory entry format in the audit_control file is incorrect.

getacdir() , getacflg() and getacna() return:

-3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SUMMARY OF TRUSTED SOLARIS CHANGES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

audit_warn(1M) , inetd(1M) , audit_control(4)

SunOS 5.7 Reference
Manual

attributes(5)

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information		
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int getacdir(char * dir, int len); int getacmin(int * min_val); int getacflg(char * auditstring, int len); int getacna(char * auditstring, int len); void setac(void); void endac(void);</pre>		
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>		
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <p>0 on success.</p> <p>-2 On failure and set <code>errno</code> to indicate the error.</p>		

getacmin() and getacflg() return:

1 On EOF .

getacdir() returns:

-1 on EOF .

2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir() .

These functions return:

-3 If the directory entry format in the audit_control file is incorrect.

getacdir() , getacflg() and getacna() return:

-3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SUMMARY OF TRUSTED SOLARIS CHANGES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

audit_warn(1M) , inetd(1M) , audit_control(4)

SunOS 5.7 Reference
Manual

attributes(5)

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information		
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int getacdir(char * dir, int len); int getacmin(int * min_val); int getacflg(char * auditstring, int len); int getacna(char * auditstring, int len); void setac(void); void endac(void);</pre>		
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <i>len</i> specifies the length of the buffer <i>dir</i>. On return, <i>dir</i> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <i>min_val</i>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <i>auditstring</i>. The parameter <i>len</i> specifies the length of the buffer <i>auditstring</i>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>		
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td> <td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td> </tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <p>0 on success.</p> <p>-2 On failure and set <code>errno</code> to indicate the error.</p>		

getacmin() and getacflg() return:

1 On EOF .

getacdir() returns:

-1 on EOF .

2 if the directory search had to start from the beginning because one of the other functions was called between calls to getacdir() .

These functions return:

-3 If the directory entry format in the audit_control file is incorrect.

getacdir() , getacflg() and getacna() return:

-3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SUMMARY OF TRUSTED SOLARIS CHANGES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

audit_warn(1M) , inetd(1M) , audit_control(4)

SunOS 5.7 Reference
Manual

attributes(5)

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char * name); struct au_class_ent *getauclassnam_r(au_class_ent_t * class_int, const char * name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t * class_int); void setauclass(void); void endauclass(void);</pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i> .</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL .</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent() .</p> <p>endauclass() may be called to indicate that audit_class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t , which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:</p> <pre>char *ac_name; au_class_t ac_class; char *ac_desc;</pre>

RETURN VALUES

getauclassnam() and getauclassnam_r() return a pointer to a struct `au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

getauclassent() and getauclassent_r() return a pointer to a struct `au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

FILES

/etc/security/audit_class Maps audit class numbers to audit class names.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except `getauclassent()` and `getauclassnam()`. The two functions, `getauclassent_r()` and `getauclassnam_r()` have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_class(4)`, `audit_event(4)`

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

All information in the MT-unsafe versions are contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char * name); struct au_class_ent *getauclassnam_r(au_class_ent_t * class_int, const char * name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t * class_int); void setauclass(void); void endauclass(void);</pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i> .</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL .</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent() .</p> <p>endauclass() may be called to indicate that audit_class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t , which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:</p> <pre>char *ac_name; au_class_t ac_class; char *ac_desc;</pre>

RETURN VALUES

getauclassnam() and getauclassnam_r() return a pointer to a struct `au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

getauclassent() and getauclassent_r() return a pointer to a struct `au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

FILES

/etc/security/audit_class Maps audit class numbers to audit class names.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except `getauclassent()` and `getauclassnam()`. The two functions, `getauclassent_r()` and `getauclassnam_r()` have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_class(4)`, `audit_event(4)`

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

All information in the MT-unsafe versions are contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char * name); struct au_class_ent *getauclassnam_r(au_class_ent_t * class_int, const char * name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t * class_int); void setauclass(void); void endauclass(void);</pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i> .</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL .</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent() .</p> <p>endauclass() may be called to indicate that audit_class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t , which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:</p> <pre>char *ac_name; au_class_t ac_class; char *ac_desc;</pre>

RETURN VALUES

getauclassnam() and getauclassnam_r() return a pointer to a struct `au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

getauclassent() and getauclassent_r() return a pointer to a struct `au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

FILES

`/etc/security/audit_class` Maps audit class numbers to audit class names.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except `getauclassent()` and `getauclassnam()`. The two functions, `getauclassent_r()` and `getauclassnam_r()` have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_class(4)`, `audit_event(4)`

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

All information in the MT-unsafe versions are contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent *getauclassnam(const char * name); struct au_class_ent *getauclassnam_r(au_class_ent_t * class_int, const char * name); struct au_class_ent *getauclassent(void); struct au_class_ent *getauclassent_r(au_class_ent_t * class_int); void setauclass(void); void endauclass(void);</pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i> .</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL .</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent() .</p> <p>endauclass() may be called to indicate that audit_class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t , which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:</p> <pre>char *ac_name; au_class_t ac_class; char *ac_desc;</pre>

RETURN VALUES

getauclassnam() and getauclassnam_r() return a pointer to a struct `au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

getauclassent() and getauclassent_r() return a pointer to a struct `au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

FILES

/etc/security/audit_class Maps audit class numbers to audit class names.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except `getauclassent()` and `getauclassnam()`. The two functions, `getauclassent_r()` and `getauclassnam_r()` have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_class(4)`, `audit_event(4)`

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

All information in the MT-unsafe versions are contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauditflags, getauditflagsbin, getauditflagschar – Convert audit flag specifications				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> int getauditflagsbin(char * auditstring, au_mask_t * masks); int getauditflagschar(char * auditstring, au_mask_t * masks, int verbose);</pre>				
DESCRIPTION	<p>getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into <i>au_mask_t</i> fields pointed to by <i>masks</i>. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in <i>audit_control(4)</i>.</p> <p>getauditflagschar() converts the <i>au_mask_t</i> fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i>. If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the text representation of the events.</p> <p><i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The <i>au_mask_t</i> fields pointed to by <i>masks</i> correspond to binary values defined in <i><bsm/audit.h></i>, which is read by <i><bsm/libbsm.h></i>.</p>				
RETURN VALUES	<p>getauditflagsbin() and getauditflagschar() return:</p> <p>0 On success.</p> <p>-1 On failure.</p>				
ATTRIBUTES	<p>See <i>attributes(5)</i> for descriptions of the following attributes:</p> <table border="1"> <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>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SUMMARY OF TRUSTED SOLARIS CHANGES	The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.				
SEE ALSO					
Trusted Solaris 7 Reference Manual	<i>audit.log(4)</i> , <i>audit_control(4)</i>				

**SunOS 5.7 Reference
Manual
BUGS**

attributes(5)

This is not a very extensible interface.

NAME	getauditflags, getauditflagsbin, getauditflagschar – Convert audit flag specifications				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> int getauditflagsbin(char * auditstring, au_mask_t * masks); int getauditflagschar(char * auditstring, au_mask_t * masks, int verbose);</pre>				
DESCRIPTION	<p>getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into <i>au_mask_t</i> fields pointed to by <i>masks</i>. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in audit_control(4).</p> <p>getauditflagschar() converts the <i>au_mask_t</i> fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i>. If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the text representation of the events.</p> <p><i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The <i>au_mask_t</i> fields pointed to by <i>masks</i> correspond to binary values defined in <bsm/audit.h>, which is read by <bsm/libbsm.h>.</p>				
RETURN VALUES	<p>getauditflagsbin() and getauditflagschar() return:</p> <p>0 On success.</p> <p>-1 On failure.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <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>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SUMMARY OF TRUSTED SOLARIS CHANGES	The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.				
SEE ALSO Trusted Solaris 7 Reference Manual	audit.log(4), audit_control(4)				

**SunOS 5.7 Reference
Manual
BUGS**

attributes(5)

This is not a very extensible interface.

NAME	getauditflags, getauditflagsbin, getauditflagschar – Convert audit flag specifications				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> int getauditflagsbin(char * auditstring, au_mask_t * masks); int getauditflagschar(char * auditstring, au_mask_t * masks, int verbose);</pre>				
DESCRIPTION	<p>getauditflagsbin() converts the character representation of audit values pointed to by <i>auditstring</i> into <i>au_mask_t</i> fields pointed to by <i>masks</i>. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in <i>audit_control(4)</i>.</p> <p>getauditflagschar() converts the <i>au_mask_t</i> fields pointed to by <i>masks</i> into a string pointed to by <i>auditstring</i>. If <i>verbose</i> is zero, the short (2-character) flag names are used. If <i>verbose</i> is non-zero, the long flag names are used. <i>auditstring</i> should be large enough to contain the text representation of the events.</p> <p><i>auditstring</i> contains a series of event names, each one identifying a single audit class, separated by commas. The <i>au_mask_t</i> fields pointed to by <i>masks</i> correspond to binary values defined in <i><bsm/audit.h></i>, which is read by <i><bsm/libbsm.h></i>.</p>				
RETURN VALUES	<p>getauditflagsbin() and getauditflagschar() return:</p> <p>0 On success.</p> <p>-1 On failure.</p>				
ATTRIBUTES	<p>See <i>attributes(5)</i> for descriptions of the following attributes:</p> <table border="1"> <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>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SUMMARY OF TRUSTED SOLARIS CHANGES	The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.				
SEE ALSO Trusted Solaris 7 Reference Manual	<i>audit.log(4)</i> , <i>audit_control(4)</i>				

**SunOS 5.7 Reference
Manual
BUGS**

attributes(5)

This is not a very extensible interface.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL .</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name .</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number .</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r() .</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t      ae_number; char          *ae_name; char
*ae_desc; au_class_t      ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY
OF TRUSTED
SOLARIS
CHANGES

SEE ALSO

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

Trusted Solaris 7 Reference Manual	getauclassent(3) , audit_class(4) , audit_event(4)
SunOS 5.7 Reference Manual	getpwnam(3C) , passwd(4) , attributes(5)
NOTES	All information for the functions <code>getauevent()</code> , <code>getauevnam()</code> , and <code>getauevnum()</code> is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char * name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char * event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t * e); struct au_event_ent *getauevnam_r(au_event_ent_t * e, char * name); struct au_event_ent *getauevnum_r(au_event_ent_t * e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t    ae_number;char                *ae_name;char
*ae_desc;au_class_t    ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY OF TRUSTED SOLARIS CHANGES SEE ALSO

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

**Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

getauclassent(3) , audit_class(4) , audit_event(4)

getpwnam(3C) , passwd(4) , attributes(5)

All information for the functions `getauevent()` , `getauevnam()` , and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL .</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name .</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number .</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonam(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r() .</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t      ae_number; char          *ae_name; char
*ae_desc; au_class_t      ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES

SEE ALSO**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

Trusted Solaris 7 Reference Manual	getauclassent(3) , audit_class(4) , audit_event(4)
SunOS 5.7 Reference Manual	getpwnam(3C) , passwd(4) , attributes(5)
NOTES	All information for the functions getauevent() , getauevnam() , and getauevnum() is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char * name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char * event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t * e); struct au_event_ent *getauevnam_r(au_event_ent_t * e, char * name); struct au_event_ent *getauevnum_r(au_event_ent_t * e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t    ae_number;char          *ae_name;char
*ae_desc;au_class_t    ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY OF TRUSTED SOLARIS CHANGES SEE ALSO

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

**Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

getauclassent(3) , audit_class(4) , audit_event(4)

getpwnam(3C) , passwd(4) , attributes(5)

All information for the functions `getauevent()` , `getauevnam()` , and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL .</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name .</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number .</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r() .</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t      ae_number; char          *ae_name; char
*ae_desc; au_class_t      ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES

SEE ALSO**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

Trusted Solaris 7 Reference Manual	getauclassent(3) , audit_class(4) , audit_event(4)
SunOS 5.7 Reference Manual	getpwnam(3C) , passwd(4) , attributes(5)
NOTES	All information for the functions getauevent() , getauevnam() , and getauevnum() is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char * name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char * event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t * e); struct au_event_ent *getauevnam_r(au_event_ent_t * e, char * name); struct au_event_ent *getauevnum_r(au_event_ent_t * e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t    ae_number;char          *ae_name;char
*ae_desc;au_class_t    ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY OF TRUSTED SOLARIS CHANGES SEE ALSO

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

**Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

getauclassent(3) , audit_class(4) , audit_event(4)

getpwnam(3C) , passwd(4) , attributes(5)

All information for the functions `getauevent()` , `getauevnam()` , and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char *name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char *event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t *e); struct au_event_ent *getauevnam_r(au_event_ent_t *e, char *name); struct au_event_ent *getauevnum_r(au_event_ent_t *e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL .</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name .</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number .</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r() .</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

RETURN VALUES

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t      ae_number; char          *ae_name; char
*ae_desc; au_class_t      ae_class;
```

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

- `/etc/security/audit_event` Maps audit event numbers to audit event names.
- `/etc/passwd` Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY
OF TRUSTED
SOLARIS
CHANGES

SEE ALSO

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

Trusted Solaris 7 Reference Manual	getauclassent(3) , audit_class(4) , audit_event(4)
SunOS 5.7 Reference Manual	getpwnam(3C) , passwd(4) , attributes(5)
NOTES	All information for the functions <code>getauevent()</code> , <code>getauevnam()</code> , and <code>getauevnum()</code> is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – Translate and verify user authorizations
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id); char auth_set_to_str(auth_set_t * authset, char separator); auth_t str_to_auth(char * auth_name); auth_set_t *str_to_auth_set(char * auth_names, char * separators); char *get_auth_text(auth_t auth_id); int chkauth(auth_t auth_id, char * username); void free_auth_set(auth_set_t * authset);</pre>
DESCRIPTION	<p>auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i> . If <i>auth_id</i> is an undefined authorization ID , the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID , TSOL_MAX_AUTH , a NULL is returned.</p> <p>auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C) . Authorization names are separated by the character <i>separator</i> . Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list.</p> <p>str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i> . <i>auth_name</i> is the name in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH .</p> <p>str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i> . <i>auth_names</i> are names in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . The string none is translated to an empty authorization list (NULL) . This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization</p>

names appear in the *auth_names* string, the function converts these invalid authorizations into *auth_id*s with the value of 0 .

`get_auth_text()` returns a pointer to the statically-allocated, null-terminated authorization description text specified by *auth_id* .

`chkauth()` returns 1 if the user, specified by *username* has the authorization specified by *auth_id* .

`free_auth_set()` releases the memory returned by `str_to_auth_set()` .

`auth_to_str()` returns a pointer to the translated authorization name string. It returns NULL on failure.

`str_to_auth()` returns the numeric authorization id. It returns 0 on failure.

`auth_set_to_str()` returns a pointer to the translated authorization names string. It returns NULL on failure.

`str_to_auth_set()` returns NULL on success.

`get_auth_text()` return a string on success and NULL upon failure.

`chkauth()` returns 1 on success and 0 on failure.

RETURN VALUES

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

NOTES

This interface is uncommitted, which means it may change between minor releases of the Trusted Solaris environment. To use these routines, the program must be loaded with the Trusted Solaris library `libtsolddb` .

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`auth_desc(4)` , `auth_name(4)`

`attributes(5)`

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent * getauusernam(const char * name); struct au_user_ent * getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent * getauusernam_r(au_user_ent_t * u, const char * name); struct au_user_ent * getauuserent_r(au_user_ent_t * u);</pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an audit_user entry.</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an audit_user entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate audit_user entries; successive calls to these functions will return either successive audit_user entries or NULL.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of audit_user entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that audit_user processing is complete; the system may then close any open audit_user file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an audit_user entry is an <code>au_user_ent</code> structure defined in <code><bsm/libbsm.h></code> with the following members:</p> <pre>char *au_name; au_mask_t au_always; au_mask_t au_never;</pre>
RETURN VALUES	<p>The <code>getauusernam()</code> function returns a pointer to an <code>au_user_ent</code> structure if it successfully locates the requested entry; otherwise it returns NULL.</p>

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****ATTRIBUTES**

The `getauuserent()` function returns a pointer to an `au_user_ent` structure if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

FILES

<code>/etc/security/audit_user</code>	Stores per-user audit event mask.
<code>/etc/passwd</code>	Stores user-id to username mappings.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_user(4)`

SunOS 5.7 Reference
Manual

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

NOTES

All information for the `getauuserent()` and `getauusername()` functions is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

The `getauusername()` and `getauuserent()` functions are not MT-safe. The `getauusername_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent * getauusernam(const char * name); struct au_user_ent * getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent * getauusernam_r(au_user_ent_t * u, const char * name); struct au_user_ent * getauuserent_r(au_user_ent_t * u);</pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry.</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or <code>NULL</code>.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code><bsm/libbsm.h></code> with the following members:</p> <pre>char *au_name; au_mask_t au_always; au_mask_t au_never;</pre>
RETURN VALUES	<p>The <code>getauusernam()</code> function returns a pointer to an <code>au_user_ent</code> structure if it successfully locates the requested entry; otherwise it returns <code>NULL</code>.</p>

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

ATTRIBUTES

The `getauuserent()` function returns a pointer to an `au_user_ent` structure if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

FILES

<code>/etc/security/audit_user</code>	Stores per-user audit event mask.
<code>/etc/passwd</code>	Stores user-id to username mappings.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_user(4)`

SunOS 5.7 Reference
Manual

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

NOTES

All information for the `getauuserent()` and `getauusernam()` functions is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

The `getauusernam()` and `getauuserent()` functions are not MT-safe. The `getauusernam_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

NAME	blportion, bcltosl, bcltoil, biltolev, getcsl, getcil, setcsl, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> bslabel_t * bcltosl(bslabel_t * label); void getcsl(bslabel_t * destination_label, const bslabel_t * source_label); void setcsl(bslabel_t * destination_label, const bslabel_t * source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() and bcltoil() provide a pointer to the sensitivity label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcsl() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcsl() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blegal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blportion, bcltosl, bcltoil, biltolev, getcs1, getcil, setcs1, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> bslabel_t * bcltosl(bslabel_t * label); void getcs1(bslabel_t * destination_label, const bslabel_t * source_label); void setcs1(bslabel_t * destination_label, const bslabel_t * source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() and bcltoil() provide a pointer to the sensitivity label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcs1() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcs1() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>b1equal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	getfauditflags – Generates the process audit state				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> int getfauditflags(au_mask_t *usremasks, au_mask_t *usrdmasks, au_mask_t *lastmasks);</pre>				
DESCRIPTION	<p>getfauditflags() generates a process audit state by combining the audit masks passed as parameters with the system audit masks specified in the audit_control(4) file. getfauditflags() obtains the system audit value by calling getacflg() (see getacinfo(3)).</p> <p><i>usremasks</i> points to au_mask_t fields which contains two values. The first value defines which events are <i>always</i> to be audited when they succeed. The second value defines which events are always to be audited when they fail.</p> <p><i>usrdmasks</i> also points to au_mask_t fields which contains two values. The first value defines which events are <i>never</i> to be audited when they succeed. The second value defines which events are never to be audited when they fail.</p> <p>The structures pointed to by <i>usremasks</i> and <i>usrdmasks</i> may be obtained from the audit_user(4) file by calling getauusernam() which returns a pointer to a structure containing all audit_user(4) fields for a user.</p> <p>The output of this function is stored in <i>lastmasks</i> which is a pointer of type au_mask_t as well. The first value defines which events are to be audited when they succeed and the second defines which events are to be audited when they fail.</p> <p>Both <i>usremasks</i> and <i>usrdmasks</i> override the values in the system audit values.</p>				
RETURN VALUES	<p>getauditflags() returns:</p> <p>0 On success.</p> <p>-1 On failure.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <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>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe.
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe.				
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.</p>				

SEE ALSO**Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual**

getacinfo(3), getauditflags(3), getauusernam(3), audit.log(4),
audit_control(4), audit_user(4)

attributes(5)

NAME	getpeerinfo – Get peer's process characteristics.
SYNOPSIS	<pre>cc [flag...] file... -lbsm -lsocket -lnsl -lintl [library...]</pre> <pre>#include <bsm/audit.h></pre>
DESCRIPTION	<p>int getpeerinfo(int <i>fd</i>, au_peerinfo_t *<i>grpinfo</i>, au_peermiscinfo_t *<i>peerinfo</i>);</p> <p>Returns the peer process' audit attributes for the peer designated by the socket or TLI file descriptor <i>fd</i>. If <i>grpinfo</i> or <i>peerinfo</i> is NULL, then the corresponding information is not obtained.</p> <p>The au_peerinfo structure has the following form:</p> <pre>struct peerinfo { ulong_t peer_ngroups /* number of elements obtained */ gid_t peer_groups[NGROUPS_UMAX]; /* peer's supplemental groups */ };</pre> <p>The remaining attributes are returned in <i>peerinfo</i> which is of type struct au_peermiscinfo and has been allocated by the calling process. The au_peermiscinfo structure has the following form:</p> <pre>struct au_peermiscinfo{ uid_t peer_ruid; /* peer's real user id */ gid_t peer_rgid; /* peer's real group id */ auditinfo_t peer_audit; /* peer's audit characteristic's */ };</pre> <p>where auditinfo_t is of type struct auditinfo which has the following form:</p> <pre>struct auditinfo{ au_id_t ai_auid; /* audit ID */ au_mask_t ai_mask; /* preselection mask */ au_tid_t ai_termid; /* audit terminal ID */ au_asid_t ai_asid; /* audit session ID */ };</pre> <p>getpeerinfo() requires that either the PRIV_PROC_AUDIT_TCB or PRIV_PROC_AUDIT_APPL privilege be asserted in a process' effective set in order to get the <i>peerinfo</i> attributes from its peer. No privileges are required to obtain just <i>grpinfo</i>.</p>
RETURN VALUES	getpeerinfo() returns 0 on success. On failure it returns a negative value and sets <i>errno</i> to indicate the error.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsl
MT-Level	MT-Safe

ERRORS

EBADF	<i>fd</i> is not a valid descriptor.
ENOTSOCK	<i>fd</i> is not a socket or TLI interface.
ENOBUFS	Insufficient resources were available in the system to perform the operation.
EADDRNOTAVAIL	Could not establish connection with server.
EINVAL	There was an internal error in which <i>fd</i> pointed to a peer process that was not recognized by its host.
ENOENT	No such port currently active on the peer.
EOPNOTSUPP	Type not <code>SOCK_DGRAM</code> or <code>SOCK_STREAM</code> , or either the local peer socket or TLI descriptor is not <code>AF_INET</code> .
EPERM	The caller does not have the proper privileges.

NOTES

Available only on Trusted Solaris systems with the auditing module enabled. The auditing module is enabled by default in the Trusted Solaris environment.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`attributes(5)`

NAME priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number

SYNOPSIS cc [*flag...*] *file...* -ltsol [*library...*]

```
#include <tsol/priv.h>
priv_t str_to_priv(const char * priv_name);

char * priv_to_str(const priv_t priv_id);

char * str_to_priv_set(const char * priv_names, priv_set_t * priv_set, const char *
separators);

char * priv_set_to_str(priv_set_t * priv_set, char separator, char * buffer, int * buflen);

char * get_priv_text(const priv_t priv_id);
```

DESCRIPTION

priv_to_str() returns a pointer to the statically allocated, null-terminated privilege name specified by *priv_id*. If *priv_id* is an undefined privilege ID , the integer ordinal of *priv_id* is returned. If *priv_id* is greater than TSOL_MAX_PRIV , the maximum allowable privilege ID , a NULL is returned.

str_to_priv() returns the numeric privilege ID specified by the null-terminated privilege name *priv_name*. Privilege names can be specified in upper or lower case. An integer ordinal in the string is also acceptable.

priv_set_to_str() appends the name of each privilege in *priv_set* to a string to which the user-supplied *buffer* of length *buflen* points. Privilege names are separated by the *separator* character. Integer ordinals name the undefined privileges found in the privilege set. String *none* identifies an empty privilege set; and *all* , a full privilege set. Privilege names in the string are sorted in alphabetical order by localized sort.

Based on the token separators (*separators*), str_to_priv_set() breaks the *priv_names* string into tokens to be translated into a privilege set. Token *none* is translated to an empty privilege set; token *all* , to a full privilege set. The presence of token *none* overrides whatever precedes it. For example, the string *file_mac_read,file_mac_write,none,proc_nofloat* produces the same result as *proc_nofloat* alone. The constructed privilege set is stored in the *priv_set_t* buffer to which *priv_set* points.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

<code>get_priv_text()</code>	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
<code>priv_to_str()</code>	Returns a pointer to the translated privilege name string. The function returns <code>NULL</code> and sets <code>errno</code> on failure.
<code>str_to_priv()</code>	Returns the numeric privilege ID. The function returns <code>-1</code> and sets <code>errno</code> on failure.
<code>priv_set_to_str()</code>	Returns a pointer to the translated privilege names string. If the passed-in <i>buflen</i> is too small to hold the string, this routine stores the required buffer size into <i>buflen</i> and returns <code>NULL</code> . The function returns <code>NULL</code> and sets <code>errno</code> on failure. This function returns <code>-1</code> if the string cannot be translated or if an integer ordinal in the string is greater than <code>TSOL_MAX_PRIV</code> .
<code>str_to_priv_set()</code>	Returns <code>NULL</code> on success. If bad privilege names appear in the <i>priv_names</i> string, the function returns a pointer to the first privilege name that is not recognizable.

ERRORS

<code>priv_to_str()</code> may fail for this reason:	
<code>EINVAL</code>	The specified <i>priv_id</i> is greater than <code>TSOL_MAX_PRIV</code> .
<code>priv_set_to_str()</code> may fail for this reason:	
<code>EFAULT</code>	The specified <i>priv_set</i> is an invalid address.
<code>str_to_priv()</code> may fail for one of these reasons:	
<code>EINVAL</code>	The specified <i>priv_name</i> does not match any of the defined privilege names.
<code>EFAULT</code>	The specified <i>priv_name</i> is an invalid address.

NOTES

To use these routines, the program must be loaded with the Trusted Solaris library `libtsol` or `libtsol.so`.

SEE ALSO

Trusted Solaris 7
Reference Manual

`priv_desc(4)` `priv_name(4)`

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profent_t *getprofentbyname(char * name, int src); profent_t *getprofent(int src); void setprofent(int stayopen , int src); void endprofent(int src); void free_profent(profent_t * profent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofentbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofent() , getprofent() , and endprofent() are used to enumerate profile entries from the database. setprofent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofent() . A call to getprofentbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofent() .</p> <p>Successive calls to getprofent() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofent() may be called to indicate that the caller expects to do no further profile entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile entry retrieval functions after calling endprofent() .</p> <p>The functions getprofentbyname() and getprofent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profent() should be used to free the pointers returned by either getprofentbyname() or getprofent() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p> <p>The parameter <i>src</i> may be set to any of TSOL_DB_SRC_FILES , TSOL_DB_SRC_NISPLUS , or TSOL_DB_SRC_SWITCH , which are defined</p>

in `<tsol/tsol.h>`. For most applications the `src` parameter should be set to `TSOL_DB_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_DB_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_DB_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofent()`, the threads will enumerate disjoint subsets of the `tsolprof` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the `struct profent_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profent_t {
    char *name;      /* name of profile */
    char *desc;      /* description */
    char *auths;     /* comma separated list of authorization numbers */
    profact_t *actions; /* linked list of actions */
    profcmd_t *cmds;  /* linked list of commands */
} profent_t;
```

The function `getprofentbyname()` returns a pointer to a `struct profent_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofent()` returns a pointer to a `struct profent_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

ERRORS

The functions `getprofentbyname()` and `getprofent()` return `NULL` on a failure.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)` ,
Notes On Multithread Applications , for information about the use of the
`_REENTRANT` flag.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

`Intro(3)`

`attributes(5)`

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profent_t * getprofentbyname(char * name, int src); profent_t * getprofent(int src); void setprofent(int stayopen, int src); void endprofent(int src); void free_profent(profent_t * profent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofentbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofent() , getprofent() , and endprofent() are used to enumerate profile entries from the database. setprofent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofent() . A call to getprofentbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofent() .</p> <p>Successive calls to getprofent() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofent() may be called to indicate that the caller expects to do no further profile entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile entry retrieval functions after calling endprofent() .</p> <p>The functions getprofentbyname() and getprofent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profent() should be used to free the pointers returned by either getprofentbyname() or getprofent() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p> <p>The parameter <i>src</i> may be set to any of TSOL_DB_SRC_FILES , TSOL_DB_SRC_NISPLUS , or TSOL_DB_SRC_SWITCH , which are defined</p>

in `<tsol/tsol.h>`. For most applications the `src` parameter should be set to `TSOL_DB_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_DB_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_DB_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofent()`, the threads will enumerate disjoint subsets of the `tsolprof` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `profent_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profent_t {
    char *name;      /* name of profile */
    char *desc;      /* description */
    char *auths;     /* comma separated list of authorization numbers */
    profact_t *actions; /* linked list of actions */
    profcmd_t *cmds;  /* linked list of commands */
} profent_t;
```

The function `getprofentbyname()` returns a pointer to a struct `profent_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofent()` returns a pointer to a struct `profent_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

ERRORS

The functions `getprofentbyname()` and `getprofent()` return `NULL` on a failure.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)` ,
Notes On Multithread Applications , for information about the use of the
`_REENTRANT` flag.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

`Intro(3)`

`attributes(5)`

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profstr_t * getprofstrbyname(char * name, int src); profstr_t * getprofstr(int src); int putprofstr(profstr_t * res, int src); int setprofstr(int stayopen, int src); int endprofstr(int src); void free_profstr(profstr_t * profstr);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the <code>tsolprof</code> NIS+ database or the <code>/etc/security/tsol/tsoluser</code> file.</p> <p><code>getprofstrbyname()</code> searches for information for a profile with the specified profile name given by the parameter <i>name</i>.</p> <p>The functions <code>setprofstr()</code>, <code>getprofstr()</code>, and <code>endprofstr()</code> are used to enumerate profile entries from the database. <code>setprofstr()</code> sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to <code>getprofstr()</code>. A call to <code>getprofstrbyname()</code> leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to <code>endprofstr()</code>.</p> <p>Successive calls to <code>getprofstr()</code> return either successive entries or return <code>NULL</code>, indicating the end of the enumeration.</p> <p><code>endprofstr()</code> may be called to indicate that the caller expects to do no further profile string retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile string retrieval functions after calling <code>endprofstr()</code>.</p> <p>The functions <code>getprofstrbyname()</code> and <code>getprofstr()</code> are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function <code>free_profstr()</code> should be used to free the pointers returned by either <code>getprofstrbyname()</code> or <code>getprofstr()</code>.</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p>

The parameter *src* may be set to any of `TSOL_SRC_FILES`, `TSOL_SRC_NISPLUS`, or `TSOL_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`. For most applications the *src* parameter should be set to `TSOL_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

The function `putprofstr()` replaces an existing profile entry or adds a new entry if the profile name does not already exist. Currently the *src* parameter is treated as `TSOL_SRC_NISPLUS`, forcing all information to be written to the NIS+ table. Use of files or of `nsswitch.conf`(4) may be supported in future releases.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofstr()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofstr()`, the threads enumerate disjoint subsets of the `tsolprof`(4) database.

User entries are represented by the struct `profstr_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profstr_t {
    char  name;      /* name of profile */
    char  desc;      /* description */
    char  auths;     /* comma separated list of authorization numbers */
    char  actions;   /* semicolon separated action descriptions */
    char  cmds;      /* semicolon separated command descriptions */
} profstr_t;
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The function `getprofstrbyname()` returns a pointer to a `profstr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofstr()` returns a pointer to a `profstr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The function `putprofstr()` returns 0 on success.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

Intro(3)

attributes(5)

Programs that use the interfaces described here cannot be linked statically since the implementation of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see Intro(3) , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profstr_t *getprofstrbyname(char * name , int src); profstr_t *getprofstr(int src); int putprofstr(profstr_t * res , int src); int setprofstr(int stayopen , int src); int endprofstr(int src); void free_profstr(profstr_t * profstr);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofstrbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofstr() , getprofstr() , and endprofstr() are used to enumerate profile entries from the database. setprofstr() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofstr() . A call to getprofstrbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofstr() .</p> <p>Successive calls to getprofstr() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofstr() may be called to indicate that the caller expects to do no further profile strry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile string retrieval functions after calling endprofstr() .</p> <p>The functions getprofstrbyname() and getprofstr() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profstr() should be used to free the pointers returned by either getprofstrbyname() or getprofstr() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p>

The parameter *src* may be set to any of `TSOL_SRC_FILES`, `TSOL_SRC_NISPLUS`, or `TSOL_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`. For most applications the *src* parameter should be set to `TSOL_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

The function `putprofstr()` replaces an existing profile entry or adds a new entry if the profile name does not already exist. Currently the *src* parameter is treated as `TSOL_SRC_NISPLUS`, forcing all information to be written to the NIS+ table. Use of files or of `nsswitch.conf`(4) may be supported in future releases.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofstr()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofstr()`, the threads enumerate disjoint subsets of the `tsolprof`(4) database.

User entries are represented by the struct `profstr_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profstr_t {
    char  name;      /* name of profile */
    char  desc;      /* description */
    char  auths;     /* comma separated list of authorization numbers */
    char  actions;   /* semicolon separated action descriptions */
    char  cmds;      /* semicolon separated command descriptions */
} profstr_t;
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The function `getprofstrbyname()` returns a pointer to a `profstr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofstr()` returns a pointer to a `profstr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The function `putprofstr()` returns 0 on success.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

Intro(3)

attributes(5)

Programs that use the interfaces described here cannot be linked statically since the implementation of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)` , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

NAME	getsockopt, setsockopt – Get and set options on sockets						
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <sys/socket.h> int getsockopt(int s, int level, int optname, void * optval, socklen_t * optlen); int setsockopt(int s, int level, int optname, const void * optval, socklen_t optlen);</pre>						
DESCRIPTION	<p>getsockopt() and setsockopt() manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost “socket” level.</p> <p>When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the “socket” level, <i>level</i> is specified as SOL_SOCKET . To manipulate options at any other level, <i>level</i> is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, <i>level</i> is set to the TCP protocol number (see getprotobyname(3N)).</p> <p>The parameters <i>optval</i> and <i>optlen</i> are used to access option values for setsockopt() . For getsockopt() , they identify a buffer in which the value(s) for the requested option(s) are to be returned. For getsockopt() , <i>optlen</i> is a value-result parameter, initially containing the size of the buffer pointed to by <i>optval</i> , and modified on return to indicate the actual size of the value returned. Use a 0 <i>optval</i> if no option value is to be supplied or returned.</p> <p><i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <sys/socket.h> contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name.</p> <p>Most socket-level options take an int for <i>optval</i> . For setsockopt() , the <i>optval</i> parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a struct linger parameter that specifies the desired state of the option and the linger interval (see below). struct linger is defined in <sys/socket.h> . struct linger contains the following members:</p> <table> <tr> <td>l_onoff</td><td>on = 1/off = 0</td></tr> <tr> <td>l_linger</td><td>linger time, in seconds</td></tr> </table> <p>The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt() .</p> <table> <tr> <td>SO_DEBUG</td><td>enable/disable recording of debugging information</td></tr> </table>	l_onoff	on = 1/off = 0	l_linger	linger time, in seconds	SO_DEBUG	enable/disable recording of debugging information
l_onoff	on = 1/off = 0						
l_linger	linger time, in seconds						
SO_DEBUG	enable/disable recording of debugging information						

SO_REUSEADDR	enable/disable local address reuse
SO_KEEPAIVE	enable/disable keep connections alive
SO_DONTROUTE	enable/disable routing bypass for outgoing messages
SO_LINGER	linger on close if data is present
SO_BROADCAST	enable/disable permission to transmit broadcast messages
SO_OOBINLINE	enable/disable reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_DGRAM_ERRIND	application wants delayed error
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules.

SO_REUSEADDR indicates that the rules used in validating addresses supplied in a `bind(3N)` call should allow reuse of local addresses. SO_KEEPAIVE enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified using a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messages are queued on a socket and a `close(2)` is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the `close()` attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the `setsockopt()` call when SO_LINGER is requested). If SO_LINGER is disabled and a `close()` is issued, the system will process the `close()` in a manner that allows the process to continue as quickly as possible.

The option SO_BROADCAST requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with `recv()` or `read()` calls without the MSG_OOB flag. No privilege is required to set the SO_BROADCAST flag, and any

user may do so; however, the `PRIV_NET_BROADCAST` privilege is required to use a broadcast address.

`SO_SNDBUF` and `SO_RCVBUF` are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. SunOS sets the maximum buffer size for both UDP and TCP to 256 Kbytes.

By default, delayed errors (such as ICMP port unreachable packets) are returned only for connected datagram sockets. `SO_DGRAM_ERRIND` makes it possible to receive errors for datagram sockets that are not connected. When this option is set, certain delayed errors received after completion of a `sendto()` or `sendmsg()` operation will cause a subsequent `sendto()` or `sendmsg()` operation using the same destination address (`to` parameter) to fail with the appropriate error. See `send(3N)`.

Finally, `SO_TYPE` and `SO_ERROR` are options used only with `getsockopt()`. `SO_TYPE` returns the type of the socket (for example, `SOCK_STREAM`). It is useful for servers that inherit sockets on startup. `SO_ERROR` returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

RETURN VALUES

`getsockopt()` returns:

0 On success.

-1 On failure, and sets `errno` to indicate the error.

ERRORS

The call succeeds unless:

<code>EBADF</code>	The argument <code>s</code> is not a valid file descriptor.
<code>ENOMEM</code>	There was insufficient memory available for the operation to complete.
<code>ENOPROTOOPT</code>	The option is unknown at the level indicated.
<code>ENOSR</code>	There were insufficient STREAMS resources available for the operation to complete.
<code>ENOTSOCK</code>	The argument <code>s</code> is not a socket.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

A process must have the `PRIV_NET_RAWACCESS` privilege in order to specify IP options 130 or 134 (`IPOPT_SEC` and `IPOPT_CIPSO` , respectively, as defined in `<inet/ip.h>`). The former refers to the Basic Security Option and the latter refers to the CIPSO option. A process must have the `PRIV_NET_BROADCAST` privilege to use a broadcast address.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`read(2)` , `bind(3N)`

**SunOS 5.7 Reference
Manual**

`close(2)` , `ioctl(2)` , `getprotobyname(3N)` , `recv(3N)` , `send(3N)` ,
`socket(3N)` , `attributes(5)`

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get Trusted Solaris user security attributes
SYNOPSIS	<pre>cc [flag...] file... -ltsolddb -ltsol -lnsl -lcmd [library...] #include <tsol/user.h> userent_t *getuserentbyname(char *user, int src); userent_t *getuserentbyuid(uid_t uid, int src); userent_t *getuserent(int src); void setuserent(int stayopen, int src); void enduserent(int src); void free_userent(userent_t *userent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user attributes from the <code>tsoluser</code> NIS+ database.</p> <p><code>getuserentbyname()</code> searches for information for a user with the specified user name <i>user</i>. <code>getuserentbyuid()</code> searches for information for a user with the specified user ID <i>uid</i>.</p> <p>The functions <code>setuserent()</code>, <code>getuserent()</code>, and <code>enduserent()</code> are used to list user entries from the database. <code>setuserent()</code> sets (or resets) the enumeration to the beginning of the set of Trusted Solaris user entries. This function should be called before the first call to <code>getuserent()</code>. A call to <code>getuserbyname()</code> or <code>getuserentbyuid()</code> leaves the list position in an indeterminate state. If the <i>stayopen</i> flag is not zero, the system may keep allocated resources such as open file descriptors until a subsequent call to <code>enduserent()</code>.</p> <p>Successive calls to <code>getuserent()</code> return either successive entries or <code>NULL</code>, which indicates the end of the list.</p> <p><code>enduserent()</code> may be called when the caller expects to do no further user-entry-retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more user-entry-retrieval functions after calling <code>enduserent()</code>.</p> <p>The functions <code>getuserentbyname()</code>, <code>getuserentbyuid()</code>, and <code>getuserent()</code> are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function <code>free_userent()</code> is used to release memory allocated by these two functions. The parameter <i>user</i>, a pointer to the user name, must be a null-terminated character string. The parameter <i>uid</i> must be a <code>uid_t</code>.</p>

The parameter *src* may be set to `TSOL_DB_SRC_FILES`, `TSOL_DB_SRC_NISPLUS`, or `TSOL_DB_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`.

For listing in multithreaded applications, the position within the list is a process-wide-property shared by all threads. `setuserent()` may be used in a multithreaded application but resets the list position for all threads. If multiple threads interleave calls to `getuserent()`, the threads will list disjoint subsets of the `tsoluser` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `userent_t` structure defined in `<tsol/user.h>`:

```
typedef struct userent_s {
    char *name;           /* user associated with this entry */
    char *lock;           /* is account locked? */
    char *badlogins;       /* how many failed login attempts so far */
    char *generation;      /* method of password generation */
    char *profiles;        /* user profiles used */
    char *roles;           /* roles assumable */
    char *idletime;        /* minutes a workstation may remain idle */
    char *idlecmd;         /* what to do at when idletime reached */
    char *labelview;       /* can user see ADMIN_HI and ADMIN_LOW labels */
    char *labeltrans;      /* process security attributes for label translation */
    char *labelmin;        /* allowed low login labels */
    char *labelmax;        /* allowed high login labels */
    char *usertype;        /* normal, admin-role, or non-admin-role */
    char *res1;            /* reserved for future use */
    char *res2;            /* reserved for future use */
    char *res3;            /* reserved for future use */
} userent_t;
```

If it successfully locates the requested entry, `getuserentbyname()` returns a pointer to a `userent_t`. If unsuccessful, `getuserentbyname()` returns `NULL`.

If it successfully lists an entry, `getuserent()` returns a pointer to a struct `userent_t`. If unsuccessful, `getuserent()` returns `NULL`, indicating the end of the list.

Upon success, `setuserent()` and `enduserent()` return 0.

The functions `getuserentbyname()`, `getuserentbyuid()`, and `getuserent()` return `NULL` on failure.

SEE ALSO

Trusted Solaris 7
Reference Manual

Intro(3)

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically because the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get Trusted Solaris user security attributes
SYNOPSIS	<pre>cc [flag...] file... -ltsolddb -ltsol -lnsl -lcmd [library...] #include <tsol/user.h> userent_t * getuserentbyname(char * user, int src); userent_t * getuserentbyuid(uid_t uid, int src); userent_t * getuserent(int src); void setuserent(int stayopen, int src); void enduserent(int src); void free_userent(userent_t * userent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user attributes from the tsoluser NIS+ database.</p> <p>getuserentbyname() searches for information for a user with the specified user name <i>user</i>. getuserentbyuid() searches for information for a user with the specified user ID <i>uid</i>.</p> <p>The functions setuserent(), getuserent(), and enduserent() are used to list user entries from the database. setuserent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris user entries. This function should be called before the first call to getuserent(). A call to getuserbyname() or getuserentbyuid() leaves the list position in an indeterminate state. If the <i>stayopen</i> flag is not zero, the system may keep allocated resources such as open file descriptors until a subsequent call to enduserent().</p> <p>Successive calls to getuserent() return either successive entries or NULL, which indicates the end of the list.</p> <p>enduserent() may be called when the caller expects to do no further user-entry-retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more user-entry-retrieval functions after calling enduserent().</p> <p>The functions getuserentbyname(), getuserentbyuid(), and getuserent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_userent() is used to release memory allocated by these two functions. The parameter <i>user</i>, a pointer to the user name, must be a null-terminated character string. The parameter <i>uid</i> must be a uid_t.</p>

The parameter `src` may be set to `TSOL_DB_SRC_FILES`, `TSOL_DB_SRC_NISPLUS`, or `TSOL_DB_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`.

For listing in multithreaded applications, the position within the list is a process-wide-property shared by all threads. `setuserent()` may be used in a multithreaded application but resets the list position for all threads. If multiple threads interleave calls to `getuserent()`, the threads will list disjoint subsets of the `tsoluser` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `userent_t` structure defined in `<tsol/user.h>`:

```
typedef struct userent_s {
    char *name;           /* user associated with this entry */
    char *lock;           /* is account locked? */
    char *badlogins;      /* how many failed login attempts so far */
    char *generation;     /* method of password generation */
    char *profiles;       /* user profiles used */
    char *roles;          /* roles assumable */
    char *idletime;       /* minutes a workstation may remain idle */
    char *idlecmd;        /* what to do at when idletime reached */
    char *labelview;      /* can user see ADMIN_HI and ADMIN_LOW labels */
    char *labeltrans;     /* process security attributes for label translation */
    char *labelmin;       /* allowed low login labels */
    char *labelmax;       /* allowed high login labels */
    char *usertype;       /* normal, admin-role, or non-admin-role */
    char *res1;           /* reserved for future use */
    char *res2;           /* reserved for future use */
    char *res3;           /* reserved for future use */
} userent_t;
```

If it successfully locates the requested entry, `getuserentbyname()` returns a pointer to a `userent_t`. If unsuccessful, `getuserentbyname()` returns `NULL`.

If it successfully lists an entry, `getuserent()` returns a pointer to a struct `userent_t`. If unsuccessful, `getuserent()` returns `NULL`, indicating the end of the list.

Upon success, `setuserent()` and `enduserent()` return 0.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

The functions `getuserentbyname()`, `getuserentbyuid()`, and `getuserent()` return `NULL` on failure.

`Intro(3)`

`attributes(5)`

Programs that use the interfaces described in this manual page cannot be linked statically because the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get Trusted Solaris user security attributes
SYNOPSIS	<pre>cc [flag...] file... -ltsolddb -ltsol -lnsl -lcmd [library...] #include <tsol/user.h> userent_t *getuserentbyname(char *user, int src); userent_t *getuserentbyuid(uid_t uid, int src); userent_t *getuserent(int src); void setuserent(int stayopen, int src); void enduserent(int src); void free_userent(userent_t *userent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user attributes from the <code>tsoluser</code> NIS+ database.</p> <p><code>getuserentbyname()</code> searches for information for a user with the specified user name <i>user</i>. <code>getuserentbyuid()</code> searches for information for a user with the specified user ID <i>uid</i>.</p> <p>The functions <code>setuserent()</code>, <code>getuserent()</code>, and <code>enduserent()</code> are used to list user entries from the database. <code>setuserent()</code> sets (or resets) the enumeration to the beginning of the set of Trusted Solaris user entries. This function should be called before the first call to <code>getuserent()</code>. A call to <code>getuserbyname()</code> or <code>getuserentbyuid()</code> leaves the list position in an indeterminate state. If the <i>stayopen</i> flag is not zero, the system may keep allocated resources such as open file descriptors until a subsequent call to <code>enduserent()</code>.</p> <p>Successive calls to <code>getuserent()</code> return either successive entries or <code>NULL</code>, which indicates the end of the list.</p> <p><code>enduserent()</code> may be called when the caller expects to do no further user-entry-retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more user-entry-retrieval functions after calling <code>enduserent()</code>.</p> <p>The functions <code>getuserentbyname()</code>, <code>getuserentbyuid()</code>, and <code>getuserent()</code> are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function <code>free_userent()</code> is used to release memory allocated by these two functions. The parameter <i>user</i>, a pointer to the user name, must be a null-terminated character string. The parameter <i>uid</i> must be a <code>uid_t</code>.</p>

The parameter *src* may be set to `TSOL_DB_SRC_FILES`, `TSOL_DB_SRC_NISPLUS`, or `TSOL_DB_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`.

For listing in multithreaded applications, the position within the list is a process-wide-property shared by all threads. `setuserent()` may be used in a multithreaded application but resets the list position for all threads. If multiple threads interleave calls to `getuserent()`, the threads will list disjoint subsets of the `tsoluser` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `userent_t` structure defined in `<tsol/user.h>`:

```
typedef struct userent_s {
    char *name;           /* user associated with this entry */
    char *lock;           /* is account locked? */
    char *badlogins;      /* how many failed login attempts so far */
    char *generation;     /* method of password generation */
    char *profiles;       /* user profiles used */
    char *roles;          /* roles assumable */
    char *idletime;       /* minutes a workstation may remain idle */
    char *idlecmd;        /* what to do at when idletime reached */
    char *labelview;      /* can user see ADMIN_HI and ADMIN_LOW labels */
    char *labeltrans;     /* process security attributes for label translation */
    char *labelmin;       /* allowed low login labels */
    char *labelmax;       /* allowed high login labels */
    char *usertype;       /* normal, admin-role, or non-admin-role */
    char *res1;           /* reserved for future use */
    char *res2;           /* reserved for future use */
    char *res3;           /* reserved for future use */
    userent_t *next;
};
```

If it successfully locates the requested entry, `getuserentbyname()` returns a pointer to a `userent_t`. If unsuccessful, `getuserentbyname()` returns `NULL`.

If it successfully lists an entry, `getuserent()` returns a pointer to a struct `userent_t`. If unsuccessful, `getuserent()` returns `NULL`, indicating the end of the list.

Upon success, `setuserent()` and `enduserent()` return 0.

The functions `getuserentbyname()`, `getuserentbyuid()`, and `getuserent()` return `NULL` on failure.

SEE ALSO

Trusted Solaris 7
Reference Manual

Intro(3)

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically because the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre> #include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file); </pre>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a utmp structure with the following members:</p> <pre> char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */ </pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre> short e_termination; /* termination status */ short e_exit; /* exit status */ </pre> <p>getutent() The <code>getutent()</code> function reads in the next entry from a utmp-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The <code>getutid()</code> function searches forward from the current point in the utmp file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The <code>getutline()</code> function searches forward from the current point in the utmp file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>

pututline() The `pututline()` function writes the supplied `utmp` structure into the `utmp` file. It uses `getutid()` to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of `pututline()` will have searched for the proper entry using one of the these functions. If so, `pututline()` will not search. If `pututline()` does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the `utmp` structure.

When called by a process that does not have an effective uid of 0 and a sensitivity label of `ADMIN_LOW`, `pututline()` invokes a program (that has the appropriate forced privileges) to verify and write the entry, since `/etc/utmpx` is normally writable only by a process with a UID of 0 and a sensitivity label of `ADMIN_LOW`. In this event, the `ut_name` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member must be a device special file and be writable by the user. If the process does not have the `PAF_TRUSTED_PATH` process attribute, all other fields in the entry are cleared.

setutent() The `setutent()` function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.

endutent() The `endutent()` function closes the currently open file.

utmpname() The `utmpname()` function allows the user to change the name of the file examined, from `/var/adm/utmp` to any other file. It is most often expected that this other file will be `/var/adm/wtmp`. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The `utmpname()` function does not open the file but closes the old file if it is currently open and saves the new file name.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, `utmpname()` returns 0. Otherwise, it returns 1.

USAGE

These functions use buffered standard I/O for input, but `pututline()` uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the `utmp` and `wtmp` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on the `getutxent(3C)` manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
----------------------------	---

<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

`pututline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututline()` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` functions, if the user has just modified those contents and passed the pointer back to `pututline()`.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file);</pre>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */short e_exit; /* exit status */</pre> <p>getutent() The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>

pututline()	<p>The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututline()</code> will have searched for the proper entry using one of the these functions. If so, <code>pututline()</code> will not search. If <code>pututline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmp</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
setutent()	The <code>setutent()</code> function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.
endutent()	The <code>endutent()</code> function closes the currently open file.
utmpname()	The <code>utmpname()</code> function allows the user to change the name of the file examined, from <code>/var/adm/utmp</code> to any other file. It is most often expected that this other file will be <code>/var/adm/wtmp</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpname()</code> function does not open the file but closes the old file if it is currently open and saves the new file name.
RETURN VALUES	A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, <code>utmpname()</code> returns 0. Otherwise, it returns 1.
USAGE	<p>These functions use buffered standard I/O for input, but <code>pututline()</code> uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the <code>utmp</code> and <code>wtmp</code> files.</p> <p>Applications should not access the <code>utmp</code> or <code>utmpx</code> database files directly, but should use the functions described on the <code>getutxent(3C)</code> manual page to interact with these files. Using these extended API s will ensure that the <code>utmp</code> and <code>utmpx</code> databases are maintained consistently.</p>
FILES	<div> <div><code>/var/adm/utmp</code></div> <div>User access and accounting information (old format)</div> </div>

<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

`pututline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututline()` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` functions, if the user has just modified those contents and passed the pointer back to `pututline()`.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre> #include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file); </pre>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a utmp structure with the following members:</p> <pre> char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */ </pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre> short e_termination; /* termination status */ short e_exit; /* exit status */ </pre> <p>getutent() The <code>getutent()</code> function reads in the next entry from a utmp-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The <code>getutid()</code> function searches forward from the current point in the utmp file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The <code>getutline()</code> function searches forward from the current point in the utmp file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>

pututline() The `pututline()` function writes the supplied `utmp` structure into the `utmp` file. It uses `getutid()` to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of `pututline()` will have searched for the proper entry using one of the these functions. If so, `pututline()` will not search. If `pututline()` does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the `utmp` structure.

When called by a process that does not have an effective uid of 0 and a sensitivity label of `ADMIN_LOW`, `pututline()` invokes a program (that has the appropriate forced privileges) to verify and write the entry, since `/etc/utmpx` is normally writable only by a process with a UID of 0 and a sensitivity label of `ADMIN_LOW`. In this event, the `ut_name` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member must be a device special file and be writable by the user. If the process does not have the `PAF_TRUSTED_PATH` process attribute, all other fields in the entry are cleared.

setutent() The `setutent()` function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.

endutent() The `endutent()` function closes the currently open file.

utmpname() The `utmpname()` function allows the user to change the name of the file examined, from `/var/adm/utmp` to any other file. It is most often expected that this other file will be `/var/adm/wtmp`. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The `utmpname()` function does not open the file but closes the old file if it is currently open and saves the new file name.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, `utmpname()` returns 0. Otherwise, it returns 1.

USAGE

These functions use buffered standard I/O for input, but `pututline()` uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the `utmp` and `wtmp` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on the `getutxent(3C)` manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
----------------------------	---

<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

`pututline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututline()` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute.

SEE ALSO

SunOS 5.7 Reference
Manual

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` functions, if the user has just modified those contents and passed the pointer back to `pututline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre> char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */ </pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre> short e_termination; /* termination status */ short e_exit; /* exit status */ </pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i>⇒<code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i>⇒<code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx()

The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp()

The `updwtmp()` function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx()

The `updwtmpx()` function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre> char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */ </pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre> short e_termination; /* termination status */ short e_exit; /* exit status */ </pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx()

The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp()

The `updwtmp()` function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx()

The `updwtmpx()` function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx()

The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp()

The `updwtmp()` function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx()

The `updwtmpx()` function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx()

The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp()

The `updwtmp()` function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx()

The `updwtmpx()` function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx()

The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp()

The `updwtmp()` function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx()

The `updwtmpx()` function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	getvfsaent, getvfsafile – Get vfstab_adjunct file entry						
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <stdio.h> #include <tsol/vfstab_adjunct.h> int getvfsaent(FILE * fp, struct vfsaent ** vp); int getvfsafile(FILE * fp, struct vfsaent ** vp, char * file);</pre>						
DESCRIPTION	<p>getvfsaent() and getvfsafile() each fill in the structure pointed to by <i>vp</i> with the broken-out fields of a line in the /etc/security/tsol/vfstab_adjunct file. Each line in the file contains a vfstab_adjunct structure, declared in the <tsol/vfstab_adjunct.h> header:</p> <pre>struct vfsaent { char *vfsa_fsname; char *vfsa_attrs; };</pre> <p>The <i>vfsa_fsname</i> contains the full pathname of the file system as listed in vfstab(4) . The <i>vfsa_attrs</i> points to the attribute string composed of keyword/value assignments of the form <i>keyword=value</i> separated by semicolons as described in vfstab_adjunct(4) .</p> <p>getvfsaent() returns a pointer to the next vfsaent structure in the file; so successive calls can be used to search the entire file. getvfsafile() searches the file referred to by <i>fp</i> until a mount point matching <i>file</i> is found.</p> <p>On successful return, the locations referred to by <i>*vp</i> , <i>*vp->vfsa_fsname</i> , and <i>*vp->vfsa_attrs</i> have been separately allocated and may be independently released by calls to free(3C) .</p> <p>Note that these routines do not open, close, or rewind the file.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>If the next entry is successfully read by getvfsaent() or a match is found with getvfsafile() , 0 is returned. If an end-of-file is encountered on reading, these functions return -1 . If an error is encountered, a value greater than 0 is returned. The possible error values are:</p> <table> <tr> <td>ENOMEM</td><td>Memory cannot be allocated for an entry.</td></tr> </table>	ENOMEM	Memory cannot be allocated for an entry.				
ENOMEM	Memory cannot be allocated for an entry.						

FILES

/etc/security/tsol/vfstab_adjunct

Attribute data file for mounting a file system in the Trusted Solaris environment.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

vfstab_adjunct(4)

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

These interfaces are uncommitted, which means that even though they are not expected to change, they may change between minor releases of the Trusted Solaris environment.

NAME	getvfsaent, getvfsafile – Get vfstab_adjunct file entry						
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <stdio.h> #include <tsol/vfstab_adjunct.h> int getvfsaent(FILE * fp, struct vfsaent ** vp); int getvfsafile(FILE * fp, struct vfsaent ** vp, char * file);</pre>						
DESCRIPTION	<p>getvfsaent() and getvfsafile() each fill in the structure pointed to by <i>vp</i> with the broken-out fields of a line in the /etc/security/tsol/vfstab_adjunct file. Each line in the file contains a vfstab_adjunct structure, declared in the <tsol/vfstab_adjunct.h> header:</p> <pre>struct vfsaent { char *vfa_fsname; char *vfa_attrs; };</pre> <p>The <i>vfa_fsname</i> contains the full pathname of the file system as listed in vfstab(4) . The <i>vfa_attrs</i> points to the attribute string composed of keyword/value assignments of the form <i>keyword=value</i> separated by semicolons as described in vfstab_adjunct(4) .</p> <p>getvfsaent() returns a pointer to the next vfsaent structure in the file; so successive calls can be used to search the entire file. getvfsafile() searches the file referred to by <i>fp</i> until a mount point matching <i>file</i> is found.</p> <p>On successful return, the locations referred to by <i>*vp</i> , <i>*vp->vfa_fsname</i> , and <i>*vp->vfa_attrs</i> have been separately allocated and may be independently released by calls to free(3C) .</p> <p>Note that these routines do not open, close, or rewind the file.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>If the next entry is successfully read by getvfsaent() or a match is found with getvfsafile() , 0 is returned. If an end-of-file is encountered on reading, these functions return -1 . If an error is encountered, a value greater than 0 is returned. The possible error values are:</p> <table> <tr> <td>ENOMEM</td><td>Memory cannot be allocated for an entry.</td></tr> </table>	ENOMEM	Memory cannot be allocated for an entry.				
ENOMEM	Memory cannot be allocated for an entry.						

FILES

/etc/security/tsol/vfstab_adjunct

Attribute data file for mounting a file system in the Trusted Solaris environment.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

vfstab_adjunct(4)

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

These interfaces are uncommitted, which means that even though they are not expected to change, they may change between minor releases of the Trusted Solaris environment.

NAME	btohex, bcltoh, bslttoh, bilttoh, bcleartoh, bcltoh_r, bslttoh_r, bilttoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char *bcltoh(const bclabel_t *label); char *bslttoh(const bslabel_t *label); char *bcleartoh(const bclear_t *clearance); char *bcltoh_r(const bclabel_t *label, char *hex); char *bslttoh_r(const bslabel_t *label, char *hex); char *bcleartoh_r(const bclear_t *clearance, char *hex); char *h_alloc(const unsigned char type); void h_free(char *hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bslttoh() and bslttoh_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bslttoh_r(), bilttoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

h_free() 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

The functions bcltoh(), bsltoh(), biltoh(), and bcleartoh() 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.

For multithreaded applications, the functions bcltoh_r(), bsltoh_r(), biltoh_r(), and bcleartoh_r() should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	hextob, htobcl, htobsl, htobil, htobclear – Convert hexadecimal string to binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int htobcl(const char *s, bclabel_t *label); int htobsl(const char *s, bslabel_t *label); int htobil(const char *s, bilabel_t *label); int htobclear(const char *s, bclear_t *clearance);</pre>						
DESCRIPTION	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl() converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0x information_label_hexadecimal_value [0x sensitivity_label_hexadecimal_value]</pre> <p>htobsl() converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>0x sensitivity_label_hexadecimal_value</pre> <p>htobclear() converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0x clearance_hexadecimal_value</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
SEE ALSO							
Trusted Solaris 7 Reference Manual	<p>atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3), bltype(3), blvalid(3), btohex(3), labelinfo(3), labelvers(3), sbldtos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**`attributes(5)`

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW` .

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]` ; however, the IL component is fixed at `ADMIN_LOW` .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	btohex, bcltoh, bsltoh, bilttoh, bcleartoh, bcltoh_r, bsltoh_r, bilttoh_r, bcleartoh_r, h_alloc, h_free – Convert binary label to hexadecimal
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...] #include <tsol/label.h> char * bcltoh(const bclabel_t * label); char * bsltoh(const bslabel_t * label); char * bcleartoh(const bclear_t * clearance); char * bcltoh_r(const bclabel_t * label, char * hex); char * bsltoh_r(const bslabel_t * label, char * hex); char * bcleartoh_r(const bclear_t * clearance, char * hex); char * h_alloc(const unsigned char type); void h_free(char * hex);</pre>
DESCRIPTION	<p>These functions convert binary labels into hexadecimal strings that represent the internal value.</p> <p>bcltoh() and bcltoh_r() convert a binary CMW label into a string of the form:</p> <p>0x <i>information_label_hexadecimal_value</i> [0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bsltoh() and bsltoh_r() convert a binary sensitivity label into a string of the form:</p> <p>[0x <i>sensitivity_label_hexadecimal_value</i>]</p> <p>bcleartoh() and bcleartoh_r() convert a binary clearance into a string of the form:</p> <p>0x <i>clearance_hexadecimal_value</i></p> <p>h_alloc() allocates memory for the hexadecimal value <i>type</i> for use by bcltoh_r(), bsltoh_r(), bilttoh_r(), and bcleartoh_r().</p> <p>Valid values for <i>type</i> are:</p> <p>SUN_CMW_ID <i>label</i> is a binary CMW label.</p> <p>SUN_SL_ID <i>label</i> is a binary sensitivity label.</p>

SUN_CLR_ID *label* is a binary clearance.

h_free() 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:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe with exceptions described in NOTES

SEE ALSO

Trusted Solaris 7
Reference Manual

atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltocolor(3), bltos(3), bltype(3), blvalid(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)

Trusted Solaris Developer's Guide

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

The functions bcltoh(), bsltoh(), biltoh(), and bcleartoh() 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.

For multithreaded applications, the functions bcltoh_r(), bsltoh_r(), biltoh_r(), and bcleartoh_r() should be used.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.

- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	hextob, htobcl, htobsl, htobil, htobclear – Convert hexadecimal string to binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int htobcl(const char * s, bclabel_t * label); int htobsl(const char * s, bslabel_t * label); int htobil(const char * s, bilabel_t * label); int htobclear(const char * s, bclear_t * clearance);</pre>						
DESCRIPTION	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl() converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0x information_label_hexadecimal_value [0x sensitivity_label_hexadecimal_value]</pre> <p>htobsl() converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>0x sensitivity_label_hexadecimal_value</pre> <p>htobclear() converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0x clearance_hexadecimal_value</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
SEE ALSO							
Trusted Solaris 7 Reference Manual	<p>atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), btohex(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	hextob, htobcl, htobsl, htobil, htobclear – Convert hexadecimal string to binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int htobcl(const char * s, bclabel_t * label); int htobsl(const char * s, bslabel_t * label); int htobil(const char * s, bilabel_t * label); int htobclear(const char * s, bclear_t * clearance);</pre>						
DESCRIPTION	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl() converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0x information_label_hexadecimal_value [0x sensitivity_label_hexadecimal_value]</pre> <p>htobsl() converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>0x sensitivity_label_hexadecimal_value</pre> <p>htobclear() converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0x clearance_hexadecimal_value</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
SEE ALSO	<p>atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3), bltype(3), blvalid(3), btohex(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES****attributes(5)**

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	hextob, htobcl, htobsl, htobil, htobclear – Convert hexadecimal string to binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int htobcl(const char * s, bclabel_t * label); int htobsl(const char * s, bslabel_t * label); int htobil(const char * s, bilabel_t * label); int htobclear(const char * s, bclear_t * clearance);</pre>						
DESCRIPTION	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl() converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0x information_label_hexadecimal_value [0x sensitivity_label_hexadecimal_value]</pre> <p>htobsl() converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>0x sensitivity_label_hexadecimal_value</pre> <p>htobclear() converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0x clearance_hexadecimal_value</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
SEE ALSO	<p>atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), btohex(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	hextob, htobcl, htobsl, htobil, htobclear – Convert hexadecimal string to binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int htobcl(const char * s, bclabel_t * label); int htobsl(const char * s, bslabel_t * label); int htobil(const char * s, bilabel_t * label); int htobclear(const char * s, bclear_t * clearance);</pre>						
DESCRIPTION	<p>These functions convert hexadecimal string representations of internal label values into binary labels.</p> <p>htobcl() converts into a binary CMW label, a hexadecimal string of the form:</p> <pre>0x information_label_hexadecimal_value [0x sensitivity_label_hexadecimal_value]</pre> <p>htobsl() converts into a binary sensitivity label, a hexadecimal string of the form:</p> <pre>0x sensitivity_label_hexadecimal_value</pre> <p>htobclear() converts into a binary clearance, a hexadecimal string of the form:</p> <pre>0x clearance_hexadecimal_value</pre>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	These functions return non-zero if the conversion was successful, otherwise zero is returned.						
SEE ALSO	<p>atohexlabel(1M), hextoalabel(1M), bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3), bltype(3), blvalid(3), btohex(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	initgroups – Initialize the supplementary group access list				
SYNOPSIS	<pre>#include <grp.h> #include <sys/types.h> int initgroups(const char *name, gid_t basegid);</pre>				
DESCRIPTION	<p>The <code>initgroups()</code> function reads the group database to get the group membership for the user specified by <i>name</i>, and initializes the supplementary group access list of the calling process (see <code>getgrnam(3C)</code> and <code>getgroups(2)</code>). The <i>basegid</i> group ID is also included in the supplementary group access list. This is typically the real group ID from the user database.</p> <p>While scanning the group database, if the number of groups, including the <i>basegid</i> entry, exceeds <code>NGROUPS_MAX</code>, subsequent group entries are ignored.</p>				
RETURN VALUES	<p><code>initgroups()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>				
ERRORS	<p>The <code>initgroups()</code> function will fail and not change the supplementary group access list if:</p> <p><code>EPERM</code> The effective user ID is not superuser.</p>				
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Unsafe				
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>To succeed, <code>initgroups()</code> must have <code>PRIV_PROC_SETID</code> in its set of effective privileges.</p>				
SEE ALSO					
Trusted Solaris 7 Reference Manual	<code>getgroups(2)</code>				
SunOS 5.7 Reference Manual	<code>getgrnam(3C)</code> , <code>attributes(5)</code>				

NAME	kstat_read, kstat_write – Read or write kstat data
SYNOPSIS	<pre>cc [flags...] file ... -lkstat [library...]</pre> <pre>#include <kstat.h></pre> <pre>kid_t kstat_read(kstat_ctl_t * kc, kstat_t * ksp, void * buf);</pre> <pre>kid_t kstat_write(kstat_ctl_t * kc, kstat_t * ksp, void * buf);</pre>
DESCRIPTION	<p>kstat_read() gets data from the kernel for the kstat pointed to by <i>ksp</i>. <i>ksp->ks_data</i> is automatically allocated (or reallocated) to be large enough to hold all of the data. <i>ksp->ks_ndata</i> is set to the number of data fields, <i>ksp->ks_data_size</i> is set to the total size of the data, and <i>ksp->ks_snaptime</i> is set to the high-resolution time at which the data snapshot was taken. If <i>buf</i> is non- NULL, the data is copied from <i>ksp->ks_data</i> into <i>buf</i>.</p> <p>kstat_write() writes data from <i>buf</i>, or from <i>ksp->ks_data</i> if <i>buf</i> is NULL, to the corresponding kstat in the kernel. kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat.</p>
RETURN VALUES	On success, kstat_read() and kstat_write() return the current kstat chain ID (KCID). On failure, they return -1 .
FILES	/dev/kstat Kernel statistics driver
SUMMARY OF TRUSTED SOLARIS CHANGES	kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat .
SEE ALSO SunOS 5.7 Reference Manual	kstat(3k) , kstat_chain_update(3k) , kstat_close(3k) , kstat_data_lookup(3k) , kstat_lookup(3k) , kstat_open(3k)

NAME	kstat_read, kstat_write – Read or write kstat data
SYNOPSIS	<pre>cc [flags...] file ... -lkstat [library...]</pre> <pre>#include <kstat.h></pre> <pre>kid_t kstat_read(kstat_ctl_t * kc, kstat_t * ksp, void * buf);</pre> <pre>kid_t kstat_write(kstat_ctl_t * kc, kstat_t * ksp, void * buf);</pre>
DESCRIPTION	<p>kstat_read() gets data from the kernel for the kstat pointed to by <i>ksp</i>. <i>ksp->ks_data</i> is automatically allocated (or reallocated) to be large enough to hold all of the data. <i>ksp->ks_ndata</i> is set to the number of data fields, <i>ksp->ks_data_size</i> is set to the total size of the data, and <i>ksp->ks_snaptime</i> is set to the high-resolution time at which the data snapshot was taken. If <i>buf</i> is non- NULL, the data is copied from <i>ksp->ks_data</i> into <i>buf</i>.</p> <p>kstat_write() writes data from <i>buf</i>, or from <i>ksp->ks_data</i> if <i>buf</i> is NULL, to the corresponding kstat in the kernel. kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat.</p>
RETURN VALUES	On success, kstat_read() and kstat_write() return the current kstat chain ID (KCID). On failure, they return -1.
FILES	/dev/kstat Kernel statistics driver
SUMMARY OF TRUSTED SOLARIS CHANGES	kstat_write() requires the PRIV_SYS_CONFIG privilege and MAC write access to /dev/kstat.
SEE ALSO SunOS 5.7 Reference Manual	kstat(3k), kstat_chain_update(3k), kstat_close(3k), kstat_data_lookup(3k), kstat_lookup(3k), kstat_open(3k)

NAME	labelbuilder, tsol_lbuild_create, tsol_lbuild_get, tsol_lbuild_set, tsol_lbuild_destroy – Create a Motif-based user interface for interactively building a valid label or clearance
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtSol [library...] #include <Dt/ModLabel.h> ModLabelData * tsol_lbuild_create(Widget widget void (*event_handler) () ok_callback lbuild_attributes extended_operation, , NULL); void *tsol_lbuild_get(ModLabelData * data, lbuild_attributes extended_operation); void tsol_lbuild_set(ModLabelData * data lbuild_attributes extended_operation, , NULL); void tsol_lbuild_destroy(ModLabelData * data);</pre>
DESCRIPTION	<p>The Label builder user interface prompts the end user for information and generates a valid CMW label, information label, sensitivity label, or clearance from the user input based on specifications in the <code>label_encodings(4)</code> file on the system where the application runs. The end user can build the label or clearance by typing a text value or by interactively choosing options.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, Label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>If the end user does not have the authorization to upgrade or downgrade labels, or if the user-built label is out of the user's accreditation range, the OK and Reset pushbuttons are grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> creates the graphical user interface and returns a pointer variable of type <code>ModLabeldata*</code> that contains information on the user interface. This information is a combination of values passed in the <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <p><code>lbuild_dialog</code> The Label builder dialog box.</p>

ok The OK pushbutton.

cancel The Cancel pushbutton.

reset The Reset pushbutton.

help The Help pushbutton.

The `tsol_lbuild_create()` parameter list takes the following values:

widget The widget from which the dialog box is created. Any Motif widget can be passed.

ok_callback A callback function that implements the behavior of the OK pushbutton on the dialog box.

..., NULL A NULL terminated list of extended operations and value pairs that define the characteristics and behavior of the Label builder dialog box.

`tsol_lbuild_destroy()` destroys the `ModLabelData` structure returned by `tsol_lbuild_create()`.

`tsol_lbuild_get()` and `tsol_lbuild_set()` access the information stored in the `ModLabelData` structure returned by `tsol_lbuild_create()`.

The following extended operations can be passed to `tsol_lbuild_create()` to build the user interface, to `tsol_lbuild_get()` to retrieve information on the user interface, and to `tsol_lbuild_set()` to change the user interface information. All extended operations are valid for `tsol_lbuild_get()`, but the `*WORK*` operations are not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` because these values are set from input supplied by the end user. These exceptions are noted in the descriptions.

LBUILD_MODE

Create a user interface to build an information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

LBUILD_MODE_SL Build a sensitivity label.

LBUILD_MODE_CMW Build a CMW label.

LBUILD_MODE_CLR Build a clearance.

LBUILD_VALUE_SL

The starting sensitivity label. This value is `ADMIN_LOW` by default and is used when the mode is `LBUILD_MODE_SL`.

LBUILD_VALUE_IL

The starting information label. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_IL .

LBUILD_VALUE_CMW

The starting CMW label. This value is ADMIN_LOW[ADMIN_LOW] by default and is used when the mode is LBUILD_MODE_CMW .

LBUILD_VALUE_CLR

The starting clearance. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_CLR .

LBUILD_USERFIELD

A character string prompt that displays at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_SHOW

Show or hide the Label builder dialog box. Value is FALSE by default.

TRUE Show the Label builder dialog box.

FALSE Hide the Label builder dialog box.

LBUILD_TITLE

A character string title that appears at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_WORK_SL

Not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` . The sensitivity label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_IL

Not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` . The information label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CMW

Not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` . The CMW label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CLR

Not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` . The clearance the end user is building. Value is updated to the end user's input

when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_X

The X position in pixels of the top-left corner of the Label buider dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_Y

The Y position in pixels of the top-left corner of the Label builder dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_LOWER_BOUND

The lowest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. This value is the user's minimum label.

LBUILD_UPPER_BOUND

The highest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. A supplied value should be within the user's accreditation range. If no value is specified, the value is the user's workspace sensitivity label, or if the executable has the `PRIV_SYS_TRANS_LABEL` privilege, the value is the user's clearance.

LBUILD_CHECK_AR

Check that the user-built label entered in the Update With field is within the user's accreditation range. A value of 1 means check, and a value of 0 means do not check. If checking is on and the label is out of range, an error message is raised to the end user.

LBUILD_VIEW

Use the internal or external label representation. Value is `LBUILD_VIEW_EXTERNAL` by default.

LBUILD_VIEW_INTERNAL Use the internal names for the highest and lowest labels in the system: `ADMIN_HIGH` and `ADMIN_LOW` .

LBUILD_VIEW_EXTERNAL Promote an `ADMIN_LOW` label to the next highest label, and demote an `ADMIN_HIGH` label to the next lowest label.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The `tsol_lbuild_get()` returns -1 if it is unable to get the value.

The `tsol_lbuild_create()` routine returns a variable of type `ModLabelData` that contains the information provided in the `tsol_lbuild_create()` input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface.

EXAMPLES

EXAMPLE 1 To create a Label Builder

```
(ModLabelData *)lbldata = tsol_lbuild_create(widget0, callback_function,
    LBUILD_MODE, LBUILD_MODE_CMW,
    LBUILD_TITLE, "Setting CMW Label",
    LBUILD_VIEW, LBUILD_VIEW_INTERNAL,
    LBUILD_X, 200,
    LBUILD_Y, 200,
    LBUILD_USERFIELD, "Pathname:",
    LBUILD_SHOW, FALSE,
    NULL);
```

EXAMPLE 2 To query the mode and display the Label Builder

These examples call the `tsol_lbuild_get()` routine to query the mode being used, and call the `tsol_lbuild_set()` routine so the Label builder dialog box displays.

```
mode = (int)tsol_lbuild_get(lbldata, LBUILD_MODE );

tsol_lbuild_set(lbldata, LBUILD_SHOW, TRUE,
    NULL);
```

EXAMPLE 3 To destroy the `ModLabelData` variable

This example destroys the `ModLabelData` variable returned in the call to `tsol_lbuild_create()`.

```
tsol_lbuild_destroy(lbldata);
```

FILES

`</usr/dt/include/Dt/ModLabel.h>`
Header file for label builder 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.

SEE ALSO

Trusted Solaris 7
Reference Manual

`label_encodings(4)`
Trusted Solaris Developer's Guide
Trusted Solaris administrator's document set
Trusted Solaris Label Administration
Trusted Solaris User's Guide

**SunOS 5.7 Reference
Manual****NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	labelclipping, Xbcltos, Xbsltos, Xbiltos, Xbcleartos – Translate a binary label and clip to the specified width
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]</pre> <pre>#include <Dt/label_clipping.h></pre> <pre>XmString Xbcltos(Display *display, const bclabel_t *cmwlabel, Dimension width, const XmFontList fontlist, const int flags);</pre> <pre>XmString Xbsltos(Display *display, const bxlabel_t *senslabel, Dimension width, const XmFontList fontlist, const int flags);</pre> <pre>XmString Xbcleartos(Display *display, const bclear_t *clearance, Dimension width, const XmFontList fontlist, const int flags);</pre>
DESCRIPTION	<p>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.</p> <p><i>display</i> The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i> The CMW label to be translated.</p> <p><i>senslabel</i> The sensitivity label to be translated.</p> <p><i>clearance</i> The clearance to be translated.</p> <p><i>width</i> 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(3) man page for more information on the clipped indicator. If the specified width is equal to the display width (<i>display</i>), the label is not truncated, but word-wrapped using a width of half the display width.</p> <p><i>fontlist</i> A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i> The value of flags indicates which words in the label_encodings(4) file are used for the translation. See the bltos(3) man page for a description of the flag values: LONG_WORDS , SHORT_WORDS , LONG_CLASSIFICATION , SHORT_CLASSIFICATION , ALL_ENTRIES , ACCESS_RELATED , VIEW_EXTERNAL , VIEW_INTERNAL ,</p>

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].

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity label, information label, or clearance translated. The compound string uses the language and fonts specified in *fontlist* and is clipped to *width*. These interfaces 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.

EXAMPLES

EXAMPLE 1 To 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.

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

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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bltos(3), sbltos(3), label_encodings(4)

Trusted Solaris Developer's Guide, *Trusted Solaris administrator's document set*, *Trusted Solaris Label Administration*, and *Trusted Solaris User's Guide*

**SunOS 5.7 Reference
Manual**

attributes(5), and XmStringDraw(3X), and FontList(3X) for information on the creation and structure of a font list.

NAME	labelinfo – Get information about the label encodings														
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>int labelinfo(struct label_info *info);</pre>														
DESCRIPTION	<p>Information about the label encodings is returned in the <code>label_info</code> structure pointed to by <i>info</i>.</p> <pre>struct label_info{ short ilabel_len; /*max information label length */ short slabel_len; /*max sensitivity label length */ short clabel_len; /*max CMW label length */ short clear_len; /*max clearance label length */ short vers_len; /*version string length */ short header_len; /*max len of banner page header */ short protect_as_len; /*max len of banner page protect as */ short caveats_len; /*max len of banner page caveats */ short channels_len; /*max len of banner page channels */ };</pre> <p>The fields in this structure have the following values:</p> <table> <tr> <td><code>ilabel_len</code></td><td>The maximum length of a character-coded information label returned when translated from a binary information label.</td></tr> <tr> <td><code>slabel_len</code></td><td>The maximum length of a character-coded sensitivity label returned when translated from a binary sensitivity label.</td></tr> <tr> <td><code>clabel_len</code></td><td>The maximum length of a character-coded CMW label returned when translated from a binary CMW label.</td></tr> <tr> <td><code>clear_len</code></td><td>The maximum length of a character-coded clearance returned when translated from a binary clearance.</td></tr> <tr> <td><code>vers_len</code></td><td>The length of the <code>label_encodings</code> file version string returned by <code>labelvers()</code>.</td></tr> <tr> <td><code>header_len</code></td><td>The maximum length of a printer banner page header string returned by <code>bcltobanner()</code>.</td></tr> <tr> <td><code>protect_as_len</code></td><td>The maximum length of a printer banner page “protect_as” string returned by <code>bcltobanner()</code>.</td></tr> </table>	<code>ilabel_len</code>	The maximum length of a character-coded information label returned when translated from a binary information label.	<code>slabel_len</code>	The maximum length of a character-coded sensitivity label returned when translated from a binary sensitivity label.	<code>clabel_len</code>	The maximum length of a character-coded CMW label returned when translated from a binary CMW label.	<code>clear_len</code>	The maximum length of a character-coded clearance returned when translated from a binary clearance.	<code>vers_len</code>	The length of the <code>label_encodings</code> file version string returned by <code>labelvers()</code> .	<code>header_len</code>	The maximum length of a printer banner page header string returned by <code>bcltobanner()</code> .	<code>protect_as_len</code>	The maximum length of a printer banner page “protect_as” string returned by <code>bcltobanner()</code> .
<code>ilabel_len</code>	The maximum length of a character-coded information label returned when translated from a binary information label.														
<code>slabel_len</code>	The maximum length of a character-coded sensitivity label returned when translated from a binary sensitivity label.														
<code>clabel_len</code>	The maximum length of a character-coded CMW label returned when translated from a binary CMW label.														
<code>clear_len</code>	The maximum length of a character-coded clearance returned when translated from a binary clearance.														
<code>vers_len</code>	The length of the <code>label_encodings</code> file version string returned by <code>labelvers()</code> .														
<code>header_len</code>	The maximum length of a printer banner page header string returned by <code>bcltobanner()</code> .														
<code>protect_as_len</code>	The maximum length of a printer banner page “protect_as” string returned by <code>bcltobanner()</code> .														

`caveats_len` The maximum length of a printer banner page caveats string returned by `bcltobanner()`.

`channels_len` The maximum length of a printer banner page channels string returned by `bcltobanner()`.

RETURN VALUES

`labelinfo()` returns:

- 1 On success.
- 1 If the `label_encodings` information is unavailable.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`bcltobanner(3)`, `bilconjoin(3)`, `blcompare(3)`, `blinset(3)`,
`blmanifest(3)`, `blminmax(3)`, `blportion(3)`, `bltcolor(3)`, `bltos(3)`,
`bltype(3)`, `blvalid(3)`, `btohex(3)`, `hextob(3)`, `labelvers(3)`, `sbltos(3)`,
`stobl(3)`, `label_encodings(4)`

Trusted Solaris Developer's Guide

**SunOS 5.7 Reference
Manual**

`attributes(5)`

WARNINGS

If the `label_encodings` file is modified after an application gets information about it, that information may be out of date.

BUGS

The `label_encodings` file is rarely updated on a running system and there is no way of informing an application that the `label_encodings` file has been modified.

NAME	labelvers – Get version of the label_encodings file						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>int labelvers(char **version, const int len);</pre>						
DESCRIPTION	<p><i>version</i> may point either to a pointer to pre-allocated memory or to the value (char *)0. If <i>version</i> points to a pointer to pre-allocated memory, then <i>len</i> indicates the size of that memory. If <i>version</i> points to the value (char *)0, memory is allocated using malloc() to contain the label_encodings file version string. The version string from the label_encodings file is copied into the allocated or pre-allocated memory.</p>						
RETURN VALUES	<p>labelvers() returns:</p> <ul style="list-style-type: none"> -1 If the label_encodings file is inaccessible. 0 If memory cannot be allocated for the return string or if the pre-allocated return string memory is insufficient to hold the string. The value of the pre-allocated string is set to the NULL string (*version[0] = '\00';). >0 If successful, the length of the version string including the NULL terminator. 						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
FILES	<p>/etc/security/tsol/label_encodings The label encodings file contains the classification names, words, constraints, and values for the defined labels of this system.</p>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), blportion(3), bltcolor(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), sbltos(3), stobl(3), label_encodings(4)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual****WARNINGS****NOTES****BUGS**

`free(3C)`, `malloc(3C)`, `attributes(5)`

If the `label_encodings` file is modified after the version string is obtained, that string may be out of date.

If memory is allocated by this routine, the caller must free memory with `free()` when the memory is no longer in use.

The `label_encodings` file is rarely updated on a running system, and there is no way of informing an application that the `label_encodings` file has been modified.

NAME	libt6 – TSIX trusted IPC library
SYNOPSIS	<pre>cc [flag...] file... -lt6 [library...] #include <tsix/t6attrs.h></pre>
DESCRIPTION	<p>libt6() constitutes the TSIX Application Program Interface (API). It is a library of routines that an application uses to control attribute transport during trusted interprocess communication. The routines in the library are recommended over the underlying system call interfaces for portability because they shield the application from operating system, communication protocol, and IPC mechanism specifics.</p> <p>The libt6() routines provide interfaces through which the trusted application:</p> <ul style="list-style-type: none"> ■ Specifies the security attributes used to label outgoing IPC messages (<i>on-message attributes</i>) and reads the on-message attributes associated with a received message. ■ Controls the security options of the endpoint used to perform trusted IPC.
Security Attributes	<p>The security attributes associated with the sending process are called on-message attributes because they are independent of the contents of the message. The TCBs decide what to do with the message based on the on-message attributes. The security attributes associated with a process, and therefore those that are used to label IPC messages, vary with the configuration of the system but must be a subset of the following attributes:</p> <p>Sensitivity Label Nationality Caveats Integrity Label TSIX Session ID Clearance Access Control List</p> <p>Effective Privileges Audit ID Process ID Additional Audit Information Process Attributes User ID Group ID Supplementary Group IDs</p> <hr/> <p>Some of these attributes imply component security policies that may not be available on some systems. Also, Information Labels (ILs) are now obsolete. See NOTES.</p> <hr/> <p>The TSIX application program interface allows trusted applications to change the on-message attributes associated with an outgoing message and retrieve the on-message attributes associated with an incoming message.</p>

**On-Message
Attribute Routines**

The on-message attribute routines affect the security attributes associated with outgoing messages or retrieve attributes associated with incoming messages. The caller specifies attributes to these routines through a `t6attr_t` control structure (defined in `<tsix/t6attrs.h>`), an opaque structure used to access sets of security attributes. The caller specifies the attributes applied to outbound messages or retrieved from incoming messages through TSIX routines. Specified attributes are copied from or written to the buffers accessible through the control structure. Any attributes not designated by the sender are supplied for outgoing messages by the underlying trusted kernel. The routines that send and retrieve on-message attributes operate on sockets or streams, generically referred to as endpoints.

`t6alloc_blk(3N)`

Allocates space for a `t6attr_t` control structure.

`t6free_blk(3N)`

Frees attribute control structure and buffers. This interface should be used in conjunction with `t6alloc_blk(3N)`, which allocates the space.

`t6dup_blk(3N)`

Given one attribute control structure, this routine allocates enough storage to hold a duplicate control structure and all attributes it references, and creates a duplicate.

`t6copy_blk(3N)`

Copies a `t6attr_t` control structure and the security attributes to which it points into a second, previously allocated `t6attr_t` structure and its previously allocated buffers.

`t6clear_blk(3N)`

Clears attributes from a `t6attr_t` control structure.

`t6cmp_blk(3N)`

Compares two `t6attr_t` control structures for equal attributes set.

`t6size_attr(3N)`

Gets the size of an attribute from the control structure.

`t6get_attr(3N)`

Gets an attribute handled by the control structure.

`t6set_attr(3N)`

Sets an attribute handled by the control structure.

`t6sendto(3N)`

Sends data and a specified set of security attributes on an endpoint.

`t6recvfrom(3N)`

	<p>Reads a network message and retrieves the security attributes associated with the data.</p> <p><code>t6peek_attr(3N)</code> Peeks ahead and returns the attributes associated with the next byte of data.</p> <p><code>t6last_attr(3N)</code> Returns the security attributes associated with the last byte of data read from the network endpoint.</p> <p><code>t6get_endpt_mask(3N)</code> Gets the endpoint mask.</p> <p><code>t6set_endpt_mask(3N)</code> Sets the endpoint mask.</p> <p><code>t6get_endpt_default(3N)</code> Gets the endpoint default security attributes.</p> <p><code>t6set_endpt_default(3N)</code> Sets the endpoint default security attributes.</p> <p><code>t6attr_query(3N)</code> Gets a mask that indicates which attributes came from templates.</p>
NETWORK ENDPOINT SECURITY OPTIONS	<p>A trusted application can manipulate a number of security options associated with the network endpoint via the following calls:</p> <p><code>t6ext_attr(3N)</code> Turns on or off the security extensions to the network endpoint. This must be called before using any other <code>libt6()</code> routines.</p> <p><code>t6new_attr(3N)</code> Specifies to the network endpoint that the receiving process is only interested in receiving attributes if they have changed since the last time it received them. This saves the overhead created by passing attributes unnecessarily with each message.</p>
INCLUDE FILES	<p>Any programs that use routines in this library must include the header files containing declarations pertinent to the routine. The synopsis section of each manual page indicates the required header files. Most routines in the library contain references to declarations defined in <code><tsix/t6attrs.h></code>. This file defines constants for attribute types to be used by various TSIX attribute library access functions, as well as constants used as parameters to the library functions.</p>
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p>

**TRUSTED
SOLARIS
SECURITY
ATTRIBUTES**

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	Trusted Solaris systems and on all other TSIX(RE) 1.1 –API compliant systems
MT-Level	MT-Safe

The Trusted Solaris environment supports the following security attributes:

Sensitivity Label
Clearance
Effective Privileges
Process Attributes
Effective User ID
Effective Group ID

Information Labels (ILs), however, are obsolete. See NOTES.

The Trusted Solaris environment also supports the following attributes as read only:

Session ID
Access Control List
Audit ID
Process ID
Additional Audit Information
Supplemental Group IDs

NOTES

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document.

Information labels (ILs) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any ILs on communications and files from systems running earlier releases as ADMIN_LOW.

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL]; however, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 has the following characteristics:

- ILs do not display in window labels; SLs (Sensitivity Labels) display alone within brackets.
- ILs do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting ILs are always ADMIN_LOW, and cannot be set on any objects.

- Options related to information labels in the `label_encodings(4)` file can be ignored:

```
Markings Name= Marks;  
Float Process Information Label;
```

NAME	listen – Listen for connections on a socket						
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h></pre> <pre>#include <sys/socket.h></pre>						
DESCRIPTION	<pre>int listen(int s, int backlog);</pre> <p>To accept connections, a socket is first created with <code>socket(3N)</code>, a backlog for incoming connections is specified with <code>listen()</code> and then the connections are accepted with <code>accept(3N)</code>. The <code>listen()</code> call applies only to sockets of type <code>SOCK_STREAM</code> or <code>SOCK_SEQPACKET</code>.</p> <p>The <i>backlog</i> parameter defines the maximum length the queue of pending connections may grow to.</p> <p>If a connection request arrives with the queue full, the client will receive an error with an indication of <code>ECONNREFUSED</code> for <code>AF_UNIX</code> sockets. If the underlying protocol supports retransmission, the connection request may be ignored so that retries may succeed. For <code>AF_INET</code> sockets, the tcp will retry the connection. If the <i>backlog</i> is not cleared by the time the tcp times out, the connect will fail with <code>ETIMEDOUT</code>.</p>						
RETURN VALUES	<pre>listen()</pre> returns: <p>0 On success.</p> <p>-1 On failure,.</p>						
ERRORS	<p>The call fails if:</p> <table> <tr> <td><code>EBADF</code></td><td>The argument <i>s</i> is not a valid file descriptor.</td></tr> <tr> <td><code>ENOTSOCK</code></td><td>The argument <i>s</i> is not a socket.</td></tr> <tr> <td><code>EOPNOTSUPP</code></td><td>The socket is not of a type that supports the operation <code>listen()</code>.</td></tr> </table>	<code>EBADF</code>	The argument <i>s</i> is not a valid file descriptor.	<code>ENOTSOCK</code>	The argument <i>s</i> is not a socket.	<code>EOPNOTSUPP</code>	The socket is not of a type that supports the operation <code>listen()</code> .
<code>EBADF</code>	The argument <i>s</i> is not a valid file descriptor.						
<code>ENOTSOCK</code>	The argument <i>s</i> is not a socket.						
<code>EOPNOTSUPP</code>	The socket is not of a type that supports the operation <code>listen()</code> .						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Safe						
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>If the calling process possesses the <code>PRIV_NET_MAC_READ</code> privilege, the endpoint will bind to an MLP; otherwise, it will bind to an SLP.</p>						
SEE ALSO							

Trusted Solaris 7 Reference Manual	accept(3N)
SunOS 5.7 Reference Manual	connect(3N), socket(3N), attributes(5), socket(5)
NOTES	There is currently no <i>backlog</i> limit.

NAME mldgetcwd – Get pathname of current working directory

SYNOPSIS `cc [flags...] file... -ltsol`

```
#include <unistd.h>
#include <tsol/mld.h>
extern char *mldgetcwd(char *buf, size_t size);
```

DESCRIPTION mldgetcwd() returns a pointer to the current directory pathname including any MLD adornments and SLD names. The value of *size* must be at least one greater than the length of the pathname to be returned.

If *buf* is not NULL, the pathname will be stored in the space pointed to by *buf*.

If *buf* is a NULL pointer, mldgetcwd() will obtain *size* bytes of space using malloc(3C). In this case, the pointer returned by mldgetcwd() may be used as the argument in a subsequent call to free().

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES mldgetcwd() returns NULL with *errno* set if *size* is not large enough, or if an error occurs in a lower-level function.

ERRORS mldgetcwd() will fail if one or more of the following are true:

EACCES	A parent directory cannot be read to get its name.
EINVAL	<i>size</i> is equal to 0.
ERANGE	<i>size</i> is greater than 0 and less than the length of the pathname plus 1.

EXAMPLES Here is a program that prints the current working directory.

```
#include unistd.h
#include stdio.h
main( )
{
    char *cwd;
    if ((cwd = mldgetcwd(NULL, 64)) == NULL) {
        perror("pwd");
        exit(2);
    }
    (void)printf("%s\n", cwd);
    return(0);
}
```

SEE ALSO

**Trusted Solaris 7
Reference Manual**

chdir(2)

**SunOS 5.7 Reference
Manual**

malloc(3C), attributes(5)

NOTES

Using chdir(2) in conjunction with mldgetcwd() can give unpredictable results.

NAME	mldstat, mldlstat – Get file status in multilevel directory				
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int mldstat(const char * path, struct stat * buf); int mldlstat(const char * path, struct stat * buf);</pre>				
DESCRIPTION	<p>mldstat() obtains file attributes similar to those that the stat(2) system call obtains except when <i>path</i> refers to a multilevel directory (MLD); in that case, mldstat() returns information about the MLD ; stat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldlstat() obtains file attributes similar to those that lstat() obtains except when the named file is an MLD ; in that case, mldlstat() returns information about the MLD ; lstat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldstat() and mldlstat() require mandatory read access to the final component of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege in its set of effective privileges.</p> <p>If the calling process does not have mandatory read access, mldstat() and mldlstat() return fixed values for some elements of the stat structure.</p> <p>See the stat(2) man page for a description of the stat structure pointed to by <i>buf</i> .</p>				
RETURN VALUES	<p>mldstat() and mldlstat() return:</p> <p>0 On success.</p> <p>-1 On failure, and set errno to indicate the error.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Async-Signal-Safe				
NOTES	<p>Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To bypass this delay, the process may assert the PRIV_PROC_NODELAY privilege.</p>				
SEE ALSO					

Trusted Solaris 7 Reference Manual	<code>link(2) , stat(2)</code>
SunOS 5.7 Reference Manual	<code>attributes(5) , stat(5)</code>

NAME	mldrealpath, mldrealpath1 – Return the canonicalized absolute pathname, including any MLD adornments and SLD names								
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <sys/param.h> #include <tsol/mld.h> char *mldrealpath(const char * path , char * resolved_path); #include <tsol/label.h> char *mldrealpath1(const char * path , char * resolved_path, const bslabel_t * sl);</pre>								
DESCRIPTION	<p>mldrealpath() expands all symbolic links and resolves references to './', '../', extra '/' characters, and MLD translations in the NULL terminated string named by <i>path</i> and stores the canonicalized absolute pathname in the buffer named by <i>resolved_path</i> . The resulting path will have no symbolic links components, nor any './', '../', nor any unadorned MLD s, nor any hidden SLD names for single level directories at the present sensitivity label.</p> <p>mldrealpath1() operates the same as mldrealpath() except the SLD name is relative to the sensitivity label <i>sl</i> . To specify a sensitivity label for an SLD 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.</p>								
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWtsu								
MT-Level	MT-Safe								
RETURN VALUES	mldrealpath() returns a pointer to the <i>resolved_path</i> on success. On failure, it returns NULL , sets <i>errno</i> to indicate the error, and places in <i>resolved_path</i> the absolute pathname of the <i>path</i> component which could not be resolved.								
ERRORS	<table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i> .</td></tr> <tr> <td>EFAULT</td><td><i>resolved_path</i> extends outside the process's allocated address space.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i> .</td></tr> <tr> <td>EINVAL</td><td><i>path</i> or <i>resolved_path</i> was NULL .</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> .	EFAULT	<i>resolved_path</i> extends outside the process's allocated address space.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	EINVAL	<i>path</i> or <i>resolved_path</i> was NULL .
EACCES	Search permission is denied for a component of the path prefix of <i>path</i> .								
EFAULT	<i>resolved_path</i> extends outside the process's allocated address space.								
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .								
EINVAL	<i>path</i> or <i>resolved_path</i> was NULL .								

EIO	An I/O error occurred while reading from or writing to the file system.
ENOENT	The named file does not exist.
ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> . A pathname component is longer than <code>NAME_MAX</code> (see <code>sysconf(3)</code>) while <code>_POSIX_NO_TRUNC</code> is in effect (see <code>pathconf(2)</code>).
<code>readlink(2)</code> , <code>getsldname(2)</code> , <code>mldgetwd(3)</code>	

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

WARNINGS

`attributes(5)`

It indirectly invokes the `readlink(2)` system call and `mldgetwd(3)` library call (for relative path names), and hence inherits the possibility of hanging due to inaccessible file system resources.

NAME	mldrealpath, mldrealpathl – Return the canonicalized absolute pathname, including any MLD adornments and SLD names								
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <sys/param.h> #include <tsol/mld.h> char *mldrealpath(const char * path , char * resolved_path); #include <tsol/label.h> char *mldrealpathl(const char * path , char * resolved_path, const bslabel_t * sl);</pre>								
DESCRIPTION	<p>mldrealpath() expands all symbolic links and resolves references to './.', '../.', extra './' characters, and MLD translations in the NULL terminated string named by <i>path</i> and stores the canonicalized absolute pathname in the buffer named by <i>resolved_path</i> . The resulting path will have no symbolic links components, nor any './.', '../.', nor any unadorned MLD s, nor any hidden SLD names for single level directories at the present sensitivity label.</p> <p>mldrealpathl() operates the same as mldrealpath() except the SLD name is relative to the sensitivity label <i>sl</i> . To specify a sensitivity label for an SLD 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.</p>								
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWtsu								
MT-Level	MT-Safe								
RETURN VALUES	mldrealpath() returns a pointer to the <i>resolved_path</i> on success. On failure, it returns NULL , sets <i>errno</i> to indicate the error, and places in <i>resolved_path</i> the absolute pathname of the <i>path</i> component which could not be resolved.								
ERRORS	<table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i> .</td></tr> <tr> <td>EFAULT</td><td><i>resolved_path</i> extends outside the process's allocated address space.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i> .</td></tr> <tr> <td>EINVAL</td><td><i>path</i> or <i>resolved_path</i> was NULL .</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> .	EFAULT	<i>resolved_path</i> extends outside the process's allocated address space.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	EINVAL	<i>path</i> or <i>resolved_path</i> was NULL .
EACCES	Search permission is denied for a component of the path prefix of <i>path</i> .								
EFAULT	<i>resolved_path</i> extends outside the process's allocated address space.								
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .								
EINVAL	<i>path</i> or <i>resolved_path</i> was NULL .								

EIO	An I/O error occurred while reading from or writing to the file system.
ENOENT	The named file does not exist.
ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> . A pathname component is longer than <code>NAME_MAX</code> (see <code>sysconf(3)</code>) while <code>_POSIX_NO_TRUNC</code> is in effect (see <code>pathconf(2)</code>).
<code>readlink(2)</code> , <code>getsldname(2)</code> , <code>mldgetwd(3)</code>	

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

WARNINGS

`attributes(5)`

It indirectly invokes the `readlink(2)` system call and `mldgetwd(3)` library call (for relative path names), and hence inherits the possibility of hanging due to inaccessible file system resources.

NAME	mldstat, mldlstat – Get file status in multilevel directory				
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int mldstat(const char * path, struct stat * buf); int mldlstat(const char * path, struct stat * buf);</pre>				
DESCRIPTION	<p>mldstat() obtains file attributes similar to those that the stat(2) system call obtains except when <i>path</i> refers to a multilevel directory (MLD); in that case, mldstat() returns information about the MLD ; stat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldlstat() obtains file attributes similar to those that lstat() obtains except when the named file is an MLD ; in that case, mldlstat() returns information about the MLD ; lstat() returns information about the single-level directory (SLD) corresponding to the label of the calling process.</p> <p>mldstat() and mldlstat() require mandatory read access to the final component of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege in its set of effective privileges.</p> <p>If the calling process does not have mandatory read access, mldstat() and mldlstat() return fixed values for some elements of the stat structure.</p> <p>See the stat(2) man page for a description of the stat structure pointed to by <i>buf</i> .</p>				
RETURN VALUES	<p>mldstat() and mldlstat() return:</p> <p>0 On success.</p> <p>-1 On failure, and set errno to indicate the error.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Async-Signal-Safe				
NOTES	<p>Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To bypass this delay, the process may assert the PRIV_PROC_NODELAY privilege.</p>				
SEE ALSO					

Trusted Solaris 7 Reference Manual	link(2) , stat(2)
SunOS 5.7 Reference Manual	attributes(5) , stat(5)

NAME	mlock, munlock – Lock or unlock pages in memory
SYNOPSIS	
Default	<pre>#include <sys/mman.h> int mlock(caddr_t addr, size_t len); int munlock(caddr_t addr, size_t len);</pre>
Standard-conforming	<pre>#include <sys/mman.h> int mlock(const void * addr, size_t len); int munlock(const void * addr, size_t len);</pre>
DESCRIPTION	<p>The <code>mlock()</code> function uses the mappings established for the address range [<i>addr</i>, <i>addr + len</i>) to identify pages to be locked in memory. If the page identified by a mapping changes, such as occurs when a copy of a writable <code>MAP_PRIVATE</code> page is made upon the first store, the lock will be transferred to the newly copied private page.</p> <p>The <code>munlock()</code> function removes locks established with <code>mlock()</code>.</p> <p>A given page may be locked multiple times by executing an <code>mlock()</code> through different mappings. That is, if two different processes lock the same page, then the page will remain locked until both processes remove their locks. However, within a given mapping, page locks do not nest – multiple <code>mlock()</code> operations on the same address in the same process will all be removed with a single <code>munlock()</code>. Of course, a page locked in one process and mapped in another (or visible through a different mapping in the locking process) is still locked in memory. This fact can be used to create applications that do nothing other than lock important data in memory, thereby avoiding page I/O faults on references from other processes in the system.</p> <p>If the mapping through which an <code>mlock()</code> has been performed is removed, an <code>munlock()</code> is implicitly performed. An <code>munlock()</code> is also performed implicitly when a page is deleted through file removal or truncation.</p> <p>Locks established with <code>mlock()</code> are not inherited by a child process after a <code>fork()</code> and are not nested.</p> <p>Attempts to <code>mlock()</code> more memory than a system-specific limit will fail.</p>
RETURN VALUES	<p><code>mlock()</code> and <code>munlock()</code> return:</p> <ul style="list-style-type: none"> 0 On success. -1 On failure, and sets <code>errno</code> to indicate the error. No changes are made to any locks in the address space of the process
ERRORS	The <code>mlock()</code> and <code>munlock()</code> functions will fail if:

EINVAL The *addr* argument is not a multiple of the page size as returned by `sysconf(3C)` .

ENOMEM Addresses in the range [*addr*, *addr + len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

ENOSYS The system does not support this memory locking interface.

EPERM The process does not have sufficient privilege.

The `mlock()` function will fail if:

EAGAIN Some or all of the memory identified by the range [*addr*, *addr + len*) could not be locked because of insufficient system resources.

USAGE

Because of the impact on system resources, the use of `mlock()` and `munlock()` is restricted to users in administrative roles with the `PRIV_SYS_CONFIG` privilege.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, `mlock()` must have `PRIV_SYS_CONFIG` in its set of effective privileges

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`fork(2)` , `plock(3C)` , `mlockall(3C)`

**SunOS 5.7 Reference
Manual**

`memcntl(2)` , `mmap(2)` , `sysconf(3C)` , `attributes(5)` , `standards(5)`

NAME	mlockall, munlockall – Lock or unlock address space
SYNOPSIS	<pre>#include <sys/mman.h> int mlockall(int flags); int munlockall(void);</pre>
DESCRIPTION	<p>The <code>mlockall()</code> function locks in memory all pages mapped by an address space.</p> <p>The value of <i>flags</i> determines whether the pages to be locked are those currently mapped by the address space, those that will be mapped in the future, or both:</p> <pre>MCL_CURRENT Lock current mappings MCL_FUTURE Lock future mappings</pre> <p>If <code>MCL_FUTURE</code> is specified for <code>mlockall()</code>, mappings are locked as they are added to the address space (or replace existing mappings), provided sufficient memory is available. Locking in this manner is not persistent across the <code>exec</code> family of functions (see <code>exec(2)</code>).</p> <p>Mappings locked using <code>mlockall()</code> with any option may be explicitly unlocked with a <code>munlock()</code> call (see <code>mlock(3C)</code>).</p> <p>The <code>munlockall()</code> function removes address space locks and locks on mappings in the address space.</p> <p>All conditions and constraints on the use of locked memory that apply to <code>mlock(3C)</code> also apply to <code>mlockall()</code>.</p> <p>Locks established with <code>mlockall()</code> are not inherited by a child process after a <code>fork(2)</code> call, and are not nested.</p>
RETURN VALUES	<p><code>mlockall()</code> and <code>munlockall()</code> return:</p> <p>0 On success.</p> <p>-1 On failure, and set <code>errno</code> to indicate the error.</p>
ERRORS	<p>The <code>mlockall()</code> and <code>munlockall()</code> functions will fail if:</p> <p><code>EAGAIN</code> Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.</p> <p><code>EINVAL</code> The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code>.</p>

EPERM The process does not have sufficient privilege.

USAGE

The `mlockall()` and `munlockall()` functions require `PRIV_SYS_CONFIG` in their set of effective privileges.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, the `mlockall()` and `munlockall()` functions require `PRIV_SYS_CONFIG` in their set of effective privileges.

SEE ALSO

Trusted Solaris 7
Reference Manual

`exec(2)` , `fork(2)` , `plock(3C)` , `mlock(3C)`

SunOS 5.7 Reference
Manual

`memcntl(2)` , `mmap(2)` , `sysconf(3C)` , `attributes(5)`

NAME	mlock, munlock – Lock or unlock pages in memory
SYNOPSIS	
Default	<pre>#include <sys/mman.h> int mlock(caddr_t addr, size_t len); int munlock(caddr_t addr, size_t len);</pre>
Standard-conforming	<pre>#include <sys/mman.h> int mlock(const void * addr, size_t len); int munlock(const void * addr, size_t len);</pre>
DESCRIPTION	<p>The <code>mlock()</code> function uses the mappings established for the address range [<code>addr</code>, <code>addr + len</code>) to identify pages to be locked in memory. If the page identified by a mapping changes, such as occurs when a copy of a writable <code>MAP_PRIVATE</code> page is made upon the first store, the lock will be transferred to the newly copied private page.</p> <p>The <code>munlock()</code> function removes locks established with <code>mlock()</code>.</p> <p>A given page may be locked multiple times by executing an <code>mlock()</code> through different mappings. That is, if two different processes lock the same page, then the page will remain locked until both processes remove their locks. However, within a given mapping, page locks do not nest – multiple <code>mlock()</code> operations on the same address in the same process will all be removed with a single <code>munlock()</code>. Of course, a page locked in one process and mapped in another (or visible through a different mapping in the locking process) is still locked in memory. This fact can be used to create applications that do nothing other than lock important data in memory, thereby avoiding page I/O faults on references from other processes in the system.</p> <p>If the mapping through which an <code>mlock()</code> has been performed is removed, an <code>munlock()</code> is implicitly performed. An <code>munlock()</code> is also performed implicitly when a page is deleted through file removal or truncation.</p> <p>Locks established with <code>mlock()</code> are not inherited by a child process after a <code>fork()</code> and are not nested.</p> <p>Attempts to <code>mlock()</code> more memory than a system-specific limit will fail.</p>
RETURN VALUES	<p><code>mlock()</code> and <code>munlock()</code> return:</p> <ul style="list-style-type: none"> 0 On success. -1 On failure, and sets <code>errno</code> to indicate the error. No changes are made to any locks in the address space of the process
ERRORS	The <code>mlock()</code> and <code>munlock()</code> functions will fail if:

EINVAL The *addr* argument is not a multiple of the page size as returned by `sysconf(3C)` .

ENOMEM Addresses in the range [*addr*, *addr + len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

ENOSYS The system does not support this memory locking interface.

EPERM The process does not have sufficient privilege.

The `mlock()` function will fail if:

EAGAIN Some or all of the memory identified by the range [*addr*, *addr + len*) could not be locked because of insufficient system resources.

USAGE

Because of the impact on system resources, the use of `mlock()` and `munlock()` is restricted to users in administrative roles with the `PRIV_SYS_CONFIG` privilege.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, `mlock()` must have `PRIV_SYS_CONFIG` in its set of effective privileges

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`fork(2)` , `plock(3C)` , `mlockall(3C)`

**SunOS 5.7 Reference
Manual**

`memcntl(2)` , `mmap(2)` , `sysconf(3C)` , `attributes(5)` , `standards(5)`

NAME	mlockall, munlockall – Lock or unlock address space
SYNOPSIS	<pre>#include <sys/mman.h> int mlockall(int flags); int munlockall(void);</pre>
DESCRIPTION	<p>The <code>mlockall()</code> function locks in memory all pages mapped by an address space.</p> <p>The value of <i>flags</i> determines whether the pages to be locked are those currently mapped by the address space, those that will be mapped in the future, or both:</p> <pre>MCL_CURRENT Lock current mappings MCL_FUTURE Lock future mappings</pre> <p>If <code>MCL_FUTURE</code> is specified for <code>mlockall()</code>, mappings are locked as they are added to the address space (or replace existing mappings), provided sufficient memory is available. Locking in this manner is not persistent across the <code>exec</code> family of functions (see <code>exec(2)</code>).</p> <p>Mappings locked using <code>mlockall()</code> with any option may be explicitly unlocked with a <code>munlock()</code> call (see <code>mlock(3C)</code>).</p> <p>The <code>munlockall()</code> function removes address space locks and locks on mappings in the address space.</p> <p>All conditions and constraints on the use of locked memory that apply to <code>mlock(3C)</code> also apply to <code>mlockall()</code>.</p> <p>Locks established with <code>mlockall()</code> are not inherited by a child process after a <code>fork(2)</code> call, and are not nested.</p>
RETURN VALUES	<p><code>mlockall()</code> and <code>munlockall()</code> return:</p> <p>0 On success.</p> <p>-1 On failure, and set <code>errno</code> to indicate the error.</p>
ERRORS	<p>The <code>mlockall()</code> and <code>munlockall()</code> functions will fail if:</p> <p><code>EAGAIN</code> Some or all of the memory in the address space could not be locked due to sufficient resources. This error condition applies to <code>mlockall()</code> only.</p> <p><code>EINVAL</code> The <i>flags</i> argument contains values other than <code>MCL_CURRENT</code> and <code>MCL_FUTURE</code>.</p>

EPERM The process does not have sufficient privilege.

USAGE

The `mlockall()` and `munlockall()` functions require `PRIV_SYS_CONFIG` in their set of effective privileges.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, the `mlockall()` and `munlockall()` functions require `PRIV_SYS_CONFIG` in their set of effective privileges.

SEE ALSO

Trusted Solaris 7
Reference Manual

`exec(2)` , `fork(2)` , `plock(3C)` , `mlock(3C)`

SunOS 5.7 Reference
Manual

`memcntl(2)` , `mmap(2)` , `sysconf(3C)` , `attributes(5)`

NAME	ftw, nftw – Walk a file tree																		
SYNOPSIS	<pre>#include <ftw.h> int ftw(const char * path, int (* fn) (const char *, const struct stat *, int), int depth); int nftw(const char * path, int (* fn) (const char *, const struct stat *, int, struct FTW*), int depth, int flags);</pre>																		
DESCRIPTION	<p>The <code>ftw()</code> function recursively descends the directory hierarchy rooted in <i>path</i>. For each object in the hierarchy, <code>ftw()</code> calls the user-defined function <i>fn</i>, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a <code>stat</code> structure (see <code>stat(2)</code>) containing information about the object, and an integer. Possible values of the integer, defined in the <code><ftw.h></code> header, are:</p> <table> <tr> <td>FTW_F</td><td>The object is a file.</td></tr> <tr> <td>FTW_D</td><td>The object is a directory.</td></tr> <tr> <td>FTW_DNR</td><td>The object is a directory that cannot be read. Descendants of the directory will not be processed.</td></tr> <tr> <td>FTW_NS</td><td>The <code>stat()</code> function failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The <code>stat</code> buffer passed to <i>fn</i> is undefined.</td></tr> </table> <p>The <code>ftw()</code> function visits a directory before visiting any of its descendants.</p> <p>The tree traversal continues until the tree is exhausted, an invocation of <i>fn</i> returns a non-zero value, or some error is detected within <code>ftw()</code> (such as an I/O error). If the tree is exhausted, <code>ftw()</code> returns 0. If <i>fn</i> returns a non-zero value, <code>ftw()</code> stops its tree traversal and returns whatever value was returned by <i>fn</i>.</p> <p><code>nftw(3C)</code> recursively descends the directory hierarchy rooted in <i>path</i>. The <i>flags</i> argument is used to control the tree walk and holds one of these values:</p> <table> <tr> <td>FTW_PHYS</td><td>Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> will follow links but will not walk down any path that crosses itself.</td></tr> <tr> <td>FTW_MOUNT</td><td>The walk will not cross a mount point.</td></tr> <tr> <td>FTW_DEPTH</td><td>All subdirectories will be visited before the directory itself.</td></tr> <tr> <td>FTW_CHDIR</td><td>The walk will change to each directory before reading it.</td></tr> <tr> <td>FTW_TSOL_MLD</td><td>In all multilevel directories (MLD s) encountered as <code>nftw()</code> walks the tree, the walk will visit single-level directories (SLD s) that are dominated by the sensitivity label of the process if the process is run without privilege. If the effective privilege set of the process includes the</td></tr> </table>	FTW_F	The object is a file.	FTW_D	The object is a directory.	FTW_DNR	The object is a directory that cannot be read. Descendants of the directory will not be processed.	FTW_NS	The <code>stat()</code> function failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The <code>stat</code> buffer passed to <i>fn</i> is undefined.	FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> will follow links but will not walk down any path that crosses itself.	FTW_MOUNT	The walk will not cross a mount point.	FTW_DEPTH	All subdirectories will be visited before the directory itself.	FTW_CHDIR	The walk will change to each directory before reading it.	FTW_TSOL_MLD	In all multilevel directories (MLD s) encountered as <code>nftw()</code> walks the tree, the walk will visit single-level directories (SLD s) that are dominated by the sensitivity label of the process if the process is run without privilege. If the effective privilege set of the process includes the
FTW_F	The object is a file.																		
FTW_D	The object is a directory.																		
FTW_DNR	The object is a directory that cannot be read. Descendants of the directory will not be processed.																		
FTW_NS	The <code>stat()</code> function failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The <code>stat</code> buffer passed to <i>fn</i> is undefined.																		
FTW_PHYS	Physical walk, does not follow symbolic links. Otherwise, <code>nftw()</code> will follow links but will not walk down any path that crosses itself.																		
FTW_MOUNT	The walk will not cross a mount point.																		
FTW_DEPTH	All subdirectories will be visited before the directory itself.																		
FTW_CHDIR	The walk will change to each directory before reading it.																		
FTW_TSOL_MLD	In all multilevel directories (MLD s) encountered as <code>nftw()</code> walks the tree, the walk will visit single-level directories (SLD s) that are dominated by the sensitivity label of the process if the process is run without privilege. If the effective privilege set of the process includes the																		

PRIV_FILE_MAC_READ and PRIV_FILE_MAC_SEARCH privileges, the walk visits all SLD s in each MLD . The file system enforces all underlying DAC policies and privilege interpretations.

If the FTW_TSOL_MLD flag is *not* specified and *path* points to an adorned MLD , the walk traverses only the SLD s of this MLD . All other MLD s encountered while walking down the tree are automatically translated to the SLD at the sensitivity label of the process even if the process is run with all privileges.

If the FTW_TSOL_MLD flag is *not* specified and *path* points to an unadorned MLD , when the walk down the tree encounters this and all other MLD s, then the function automatically translates to the SLD at the sensitivity label of the process.

If the FTW_TSOL_MLD flag is *not* specified and *path* does not point to an MLD , when the walk down the tree encounters any MLD s, then the function automatically translates to the SLD at the sensitivity label of the process even if the process is run with all privileges.

The `nftw()` function calls *fn* with four arguments at each file and directory. The first argument is the pathname of the object, the second is a pointer to the `stat` structure [see `stat(2)`] containing information about the object, the third is an integer giving additional information, and the fourth is a `struct FTW` that contains the following members:

```
int    base;
int    level;
```

The `base` member is the offset into the pathname of the base name of the object. The `level` member indicates the depth relative to the rest of the walk, where the root level is zero.

The values of the third argument are as follows:

FTW_F	The object is a file.
FTW_D	The object is a directory.
FTW_DP	The object is a directory and subdirectories have been visited.
FTW_SL	The object is a symbolic link.
FTW_SLN	The object is a symbolic link that points to a non-existent file.

FTW_DNR	The object is a directory that cannot be read.]The user-defined function <i>fn</i> will not be called for any of its descendants.
FTW_NS	The <code>stat()</code> function failed on the object because of lack of appropriate permission. The <code>stat</code> buffer passed to <i>fn</i> is undefined. The <code>stat()</code> function failed for a reason other than lack of appropriate permission. <code>EACCES</code> is considered an error and <code>nftw()</code> will return <code>-1</code> .

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error (such as an I/O error) is detected within `nftw()` . If the tree is exhausted, `nftw()` returns zero. If *fn* returns a nonzero value, `nftw()` stops its tree traversal and returns whatever value *fn* returned.

Both `ftw()` and `nftw()` use one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were `1` . It must not be greater than the number of file descriptors currently available for use. The `ftw()` function will run faster if *depth* is at least as large as the number of levels in the tree. When `ftw()` and `nftw()` return, they close any file descriptors they have opened; they do not close any file descriptors that may have been opened by *fn* .

RETURN VALUES

`ftw()` and `nftw()` return:

0	On success.
-1	If either function detects an error other than <code>EACCES</code> , and sets <code>errno</code> to indicate the error.

USAGE

Because `ftw()` is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

The `ftw()` function uses `malloc(3C)` to allocate dynamic storage during its operation. If `ftw()` is forcibly terminated, such as by `longjmp(3C)` being executed by *fn* or an interrupt routine, `ftw()` will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a non-zero value at its next invocation.

The `ftw()` and `nftw()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)` .

The `ftw()` function is safe in multithreaded applications. The `nftw()` function is safe in multithreaded applications when the `FTW_CHDIR` flag is not set.

There are two versions of `nftw()` . The Solaris version, which does not traverse multilevel directories (`MLD` s), is located in `libc` ; the Trusted Solaris version,

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****ATTRIBUTES**

which traverses MLD s, is located in `libtsol`. To use the Trusted Solaris version of `nftw()`, make sure that the application uses the version of `nftw` located in `libtsol`.

The `libc` versions of `ftw()` and `nftw()` are unchanged. The Trusted Solaris version of `nftw()`, which has the additional flag `FTW_TSOL_MLD`, is available in `libtsol`. You must be careful of the library sequence when linking.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe with exceptions.

SEE ALSO
Trusted Solaris 7
Reference Manual**SunOS 5.7 Reference**
Manual

`stat(2)`

`longjmp(3C)`, `malloc(3C)`, `attributes(5)`, `lf64(5)`

NAME	<code>nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult</code> – NIS+ namespace functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result * nis_lookup(nis_name <i>name</i>, uint_t <i>flags</i>); nis_result * nis_add(nis_name <i>name</i>, nis_object * <i>obj</i>); nis_result * nis_remove(nis_name <i>name</i>, nis_object * <i>obj</i>); nis_result * nis_modify(nis_name <i>name</i>, nis_object * <i>obj</i>); void nis_freeresult(nis_result * <i>result</i>);</pre>
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3N)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3N)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in <code>nis_subr(3N)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with <code>nis_clone_object(3N)</code> (see <code>nis_subr(3N)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> takes two parameters, the name of the object to be resolved in <i>name</i>, and a <i>flags</i> parameter, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see <code>nis_tables(3N)</code>). The <code>nis_lookup()</code> function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically OR ing zero or more flags from the following list.</p>

FOLLOW_LINKS	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until the either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.
NO_CACHE	When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.
MASTER_ONLY	When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3N)</code>) which uses the environment variable <code>NIS_PATH</code> .

The status value may be translated to ascii text using the function `nis_sperrno()` [see `nis_error(3N)`].

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function `nis_add()` will take the object *obj* and add it to the NIS+ namespace with the name *name* . This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function `nis_remove()` will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not `NULL`, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the `NIS_NOTSAMEOBJ` error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function `nis_modify()` will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error `NIS_NOTSAMEOBJ` if the object identifier of the passed object does not match that of the object being modified in the namespace.

Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-null the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted `'.'`(dot) characters. If these conditions are not met the operation will fail and return the `NIS_BADNAME` error code.

Results

These functions return a pointer to a structure of type `nis_result`:

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` (see `nis_error(3N)`).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to `nis_freeresult()`. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` (see `nis_server(3N)`). Refer to `nis_objects(3N)` for a description of the *nis_object* structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

- zticks* The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
- dticks* The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
- aticks* The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
- cticks* The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

`NIS_SUCCESS`

The request was successful.

`NIS_S_SUCCESS`

The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag `NO_CACHE` when you call the lookup function.

`NIS_NOTFOUND`

The named object does not exist in the namespace.

`NIS_CACHEEXPIRED`

The object returned came from an object cache that has *expired* . The time to live value has gone to zero and the object may have changed. If the flag `NO_CACHE` was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

`NIS_NAMEUNREACHABLE`

A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the `HARD_LOOKUP` flag.

NIS_UNKNOWNOBJ

The object returned is of an unknown type.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. For the *add* , *remove* , and *modify* operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of `nis_list()` if the client specifies a callback and the server does not have enough resources to handle the callback.

NIS_SYSTEMERROR

A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.

NIS_NOT_ME

A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.

NIS_NOTMASTER

An attempt was made to update the database on a replica server.

NIS_INVALIDOBJ

The object pointed to by *obj* is not a valid NIS+ object.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_LINKNAMEERROR

The name passed resolved to a *LINK* type object and the contents of the link pointed to an invalid name.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_MODFAIL

The attempted modification failed.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type `DIRECTORY`, which contains the type of namespace and contact information for a server within that namespace.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

ENVIRONMENT VARIABLES

NIS_PATH If the flag `EXPAND_NAME` is set, this variable is the search path used by `nis_lookup()`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_add()`, `nis_modify()`, and `nis_remove()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_server(3N)`, `nis_tables(3N)`

SunOS 5.7 Reference
Manual

`nis_error(3N)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.

NAME	<code>nis_tables</code> , <code>nis_list</code> , <code>nis_add_entry</code> , <code>nis_remove_entry</code> , <code>nis_modify_entry</code> , <code>nis_first_entry</code> , <code>nis_next_entry</code> – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result *nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result *nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_first_entry(nis_name table_name); nis_result *nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[''] characters. Indexed names have the following form:</p> <pre>[colname=value, . . .], tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.</p> <p>FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3N)</code>), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <i>name</i> does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.
NO_CACHE	This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.
MASTER_ONLY	This flag is even stronger than NO_CACHE in that it specifies that the client library should <i>only</i> get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling <code>nis_getnames()</code> [see <code>nis_local_names(3N)</code>] which uses the environment variable <code>NIS_PATH</code> .

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, *callback*, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See NOTES.

`nis_add_entry()` will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_add_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the *flags* variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table. To succeed, `nis_remove_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_modify_entry()` modifies an object identified by *name*. The parameter *object* should point to an entry with the `LEN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* : *zo_owner* , *zo_group* , and *zo_access* .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag *MOD_SAMEOBJ* then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the *NIS_NOTSAMEOBJ* error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag *RETURN_RESULT* has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, *nis_modify_entry()* must inherit the *PAF_TRUSTED_PATH* attribute.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the *nis_result* structure must be copied by the caller into local storage and passed as an argument to *nis_next_entry()* .

nis_next_entry() retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to *nis_next_entry()* there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error *NIS_CHAINBROKEN* is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type *nis_result* :

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` [see `nis_error(3N)`].

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` [see `nis_names(3N)`]. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` [see `nis_server(3N)`].

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE

The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST

A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED

The entry returned came from an object cache that has *expired*. This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_CBERROR

An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS

Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ

The object pointed to by *object* is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.

NIS_LINKNAMEERROR

The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL

The attempted modification failed for some reason.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.

NIS_NAMEUNREACHABLE

This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.

NIS_NOCALLBACK

The server was unable to contact the callback service on your machine. This results in no data being returned.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_NOT_ME

A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.

NIS_NOTFOUND

No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the `nis_remove()`.

If the `FOLLOW_PATH` flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.

NIS_NOTMASTER

A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the `/var/nis/NIS_SHARED_DIRCACHE` file will need to have their cache managers restarted (use `nis_cachemgr -i`) to flush this cache.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOTSEARCHABLE

The table name resolved to a NIS+ object that was not searchable.

NIS_PARTIAL

This result is similar to `NIS_NOTFOUND` except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.

- NIS_RPCERROR
This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.
- NIS_S_NOTFOUND
The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.
- NIS_S_SUCCESS
Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.
- NIS_SUCCESS
The request was successful.
- NIS_SYSTEMERROR
Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.
- NIS_TOOMANYATTRS
The search criteria passed to the server had more attributes than the table had searchable columns.
- NIS_TRYAGAIN
The server connected to was too busy to handle your request.
`add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.
- NIS_TYPEMISMATCH
An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

ENVIRONMENT
VARIABLES

ATTRIBUTES

NIS_PATH When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`nis_cachemgr(1M)`, `nis_names(3N)`, `nis_server(3N)`,
`rpc_svc_calls(3N)`

**SunOS 5.7 Reference
Manual**

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3)`,
`nis_clone_object(3N)`, `nis_db(3N)`, `nis_objects(3N)`,
`nis_destroy_object(3N)`, `nis_error(3N)`, `nis_getnames(3N)`,
`nis_local_names(3N)`, `nis_objects(3N)`, `attributes(5)`

WARNINGS

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

NOTES

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling `nis_list()` with a callback from within a list callback function is not currently supported.

There are currently no known methods for `nis_first_entry()` and `nis_next_entry()` to get their answers from only the master server.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3N)`. Otherwise, this function is MT-Safe.

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.`, with members `wickedwitch.west.oz.` and `*.monkeys.west.oz.`, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.`, with members `toto.kansas`, `@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

NOTES

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_groups` man page refers to Solaris man pages only.

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
    (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	nis_ping, nis_checkpoint – Misc NIS+ log administration functions				
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]</pre> <pre>#include <rpcsvc/nis.h></pre> <pre>void nis_ping(nis_name <i>dirname</i>, uint32_t <i>utime</i>, nis_object * <i>dirobj</i>);</pre> <pre>nis_result * nis_checkpoint(nis_name <i>dirname</i>);</pre>				
DESCRIPTION	<p>nis_ping() is called by the master server for a directory when a change has occurred within that directory. The parameter <i>dirname</i> identifies the directory with the change. If the parameter <i>dirobj</i> is <code>NULL</code>, this function looks up the directory object for <i>dirname</i> and uses the list of replicas it contains. The parameter <i>utime</i> contains the timestamp of the last change made to the directory. This timestamp is used by the replicas when retrieving updates made to the directory.</p> <p>The effect of calling nis_ping() is to schedule an update on the replica. A short time after a ping is received, typically about two minutes, the replica compares the last update time for its databases to the timestamp sent by the ping. If the ping timestamp is later, the replica establishes a connection with the master server and request all changes from the log that occurred after the last update that it had recorded in its local log.</p> <p>To succeed, nis_ping() must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>nis_checkpoint() is used to force the service to checkpoint information that has been entered in the log but has not been checkpointed to disk. When called, this function checkpoints the database for each table in the directory, the database containing the directory and the transaction log. Care should be used in calling this function since directories that have seen a lot of changes may take several minutes to checkpoint. During the checkpointing process, the service will be unavailable for updates for all directories that are served by this machine as master.</p> <p>nis_checkpoint() returns a pointer to a <i>nis_result</i> structure (described in nis_tables(3N)). This structure should be freed with nis_freeresult() (see nis_names(3N)). The only items of interest in the returned result are the status value and the statistics.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <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>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual**

To succeed, `nis_ping()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3N)` , `nis_tables(3N)`

`nislog(1M)` , `nisfiles(4)` , `attributes(5)`

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.`, with members `wickedwitch.west.oz.`

and `*.monkeys.west.oz.`, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.`, with members `toto.kansas`,

`@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

The `nis_groups` man page refers to Solaris man pages only.

SunOS 5.7 Reference
Manual

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

NOTES

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
    (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry - NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.`, with members `wickedwitch.west.oz.` and `*.monkeys.west.oz.`, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.`, with members `toto.kansas`, `@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

NOTES

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_groups` man page refers to Solaris man pages only.

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
    (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	<code>nis_tables</code> , <code>nis_list</code> , <code>nis_add_entry</code> , <code>nis_remove_entry</code> , <code>nis_modify_entry</code> , <code>nis_first_entry</code> , <code>nis_next_entry</code> – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result *nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result *nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_first_entry(nis_name table_name); nis_result *nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[''] characters. Indexed names have the following form:</p> <pre>[colname=value, . . .], tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the tablename should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.</p> <p>FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3N)</code>), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <i>name</i> does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.
NO_CACHE	This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.
MASTER_ONLY	This flag is even stronger than NO_CACHE in that it specifies that the client library should <i>only</i> get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling <code>nis_getnames()</code> [see <code>nis_local_names(3N)</code>] which uses the environment variable <code>NIS_PATH</code> .

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, *callback*, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See **NOTES**.

`nis_add_entry()` will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_add_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the *flags* variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table. To succeed, `nis_remove_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_modify_entry()` modifies an object identified by *name*. The parameter *object* should point to an entry with the `LEN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* : *zo_owner* , *zo_group* , and *zo_access* .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag *MOD_SAMEOBJ* then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the *NIS_NOTSAMEOBJ* error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag *RETURN_RESULT* has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, *nis_modify_entry()* must inherit the *PAF_TRUSTED_PATH* attribute.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the *nis_result* structure must be copied by the caller into local storage and passed as an argument to *nis_next_entry()* .

nis_next_entry() retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to *nis_next_entry()* there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error *NIS_CHAINBROKEN* is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type *nis_result* :

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` [see `nis_error(3N)`].

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` [see `nis_names(3N)`]. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` [see `nis_server(3N)`].

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE

The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST

A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED

The entry returned came from an object cache that has *expired*. This means that the time to live value has gone to zero and the entry may have changed. If the flag `NO_CACHE` was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_CBERROR

An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS

Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type `DIRECTORY`. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ

The object pointed to by *object* is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.

NIS_LINKNAMEERROR

The name passed resolved to a *LINK* type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL

The attempted modification failed for some reason.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.

NIS_NAMEUNREACHABLE

This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the `HARD_LOOKUP` flag.

NIS_NOCALLBACK

The server was unable to contact the callback service on your machine. This results in no data being returned.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_NOT_ME

A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.

NIS_NOTFOUND

No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the `nis_remove()`.

If the `FOLLOW_PATH` flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.

NIS_NOTMASTER

A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the `/var/nis/NIS_SHARED_DIRCACHE` file will need to have their cache managers restarted (use `nis_cachemgr -i`) to flush this cache.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOTSEARCHABLE

The table name resolved to a NIS+ object that was not searchable.

NIS_PARTIAL

This result is similar to `NIS_NOTFOUND` except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.

- NIS_RPCERROR
This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.
- NIS_S_NOTFOUND
The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.
- NIS_S_SUCCESS
Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.
- NIS_SUCCESS
The request was successful.
- NIS_SYSTEMERROR
Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.
- NIS_TOOMANYATTRS
The search criteria passed to the server had more attributes than the table had searchable columns.
- NIS_TRYAGAIN
The server connected to was too busy to handle your request. `add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.
- NIS_TYPEMISMATCH
An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

ENVIRONMENT
VARIABLES

ATTRIBUTES

NIS_PATH When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

SUMMARY OF TRUSTED SOLARIS CHANGES	To succeed, <code>nis_add_entry()</code> , <code>nis_remove_entry()</code> , and <code>nis_modify_entry()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.
SEE ALSO Trusted Solaris 7 Reference Manual	<code>nis_cachemgr(1M)</code> , <code>nis_names(3N)</code> , <code>nis_server(3N)</code> , <code>rpc_svc_calls(3N)</code>
SunOS 5.7 Reference Manual	<code>niscat(1)</code> , <code>niserror(1)</code> , <code>nismatch(1)</code> , <code>syslog(3)</code> , <code>nis_clone_object(3N)</code> , <code>nis_db(3N)</code> , <code>nis_objects(3N)</code> , <code>nis_destroy_object(3N)</code> , <code>nis_error(3N)</code> , <code>nis_getnames(3N)</code> , <code>nis_local_names(3N)</code> , <code>nis_objects(3N)</code> , <code>attributes(5)</code>
WARNINGS	Use the flag <code>HARD_LOOKUP</code> carefully since it can cause the application to block indefinitely during a network partition.
NOTES	<p>The path used when the flag <code>FOLLOW_PATH</code> is specified, is the one present in the <i>first</i> table searched. The path values in tables that are subsequently searched are ignored.</p> <p>It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling <code>nis_list()</code> with a callback from within a list callback function is not currently supported.</p> <p>There are currently no known methods for <code>nis_first_entry()</code> and <code>nis_next_entry()</code> to get their answers from only the master server.</p> <p>The <code>nis_list()</code> function is not MT-Safe with callbacks. <code>nis_list()</code> callbacks are serialized. A call to <code>nis_list()</code> with a callback from within <code>nis_list()</code> will deadlock. <code>nis_list()</code> with a callback cannot be called from an rpc server. See <code>rpc_svc_calls(3N)</code>. Otherwise, this function is MT-Safe.</p>

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]</pre> <pre>#include <rpcsvc/nis.h> nis_result *nis_lookup(nis_name name, uint_t flags); nis_result *nis_add(nis_name name, nis_object *obj); nis_result *nis_remove(nis_name name, nis_object *obj); nis_result *nis_modify(nis_name name, nis_object *obj); void nis_freeresult(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3N)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3N)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in <code>nis_subr(3N)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with <code>nis_clone_object(3N)</code> (see <code>nis_subr(3N)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> takes two parameters, the name of the object to be resolved in <i>name</i>, and a flags parameter, <i>flags</i>, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see <code>nis_tables(3N)</code>). The <code>nis_lookup()</code> function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically OR ing zero or more flags from the following list.</p>

FOLLOW_LINKS	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.
NO_CACHE	When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.
MASTER_ONLY	When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3N)</code>) which uses the environment variable <code>NIS_PATH</code> .

The status value may be translated to ascii text using the function `nis_sperrno()` [see `nis_error(3N)`].

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function `nis_add()` will take the object *obj* and add it to the NIS+ namespace with the name *name* . This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function `nis_remove()` will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not `NULL`, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the `NIS_NOTSAMEOBJ` error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function `nis_modify()` will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error `NIS_NOTSAMEOBJ` if the object identifier of the passed object does not match that of the object being modified in the namespace.

Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-null the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted '.'(dot) characters. If these conditions are not met the operation will fail and return the `NIS_BADNAME` error code.

Results

These functions return a pointer to a structure of type `nis_result`:

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` (see `nis_error(3N)`).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to `nis_freeresult()`. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` (see `nis_server(3N)`). Refer to `nis_objects(3N)` for a description of the *nis_object* structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

- zticks* The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
- dticks* The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
- aticks* The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
- cticks* The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

`NIS_SUCCESS`

The request was successful.

`NIS_S_SUCCESS`

The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag `NO_CACHE` when you call the lookup function.

`NIS_NOTFOUND`

The named object does not exist in the namespace.

`NIS_CACHEEXPIRED`

The object returned came from an object cache that has *expired* . The time to live value has gone to zero and the object may have changed. If the flag `NO_CACHE` was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

`NIS_NAMEUNREACHABLE`

A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the `HARD_LOOKUP` flag.

NIS_UNKNOWNOBJ

The object returned is of an unknown type.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. For the *add* , *remove* , and *modify* operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of `nis_list()` if the client specifies a callback and the server does not have enough resources to handle the callback.

NIS_SYSTEMERROR

A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.

NIS_NOT_ME

A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.

NIS_NOTMASTER

An attempt was made to update the database on a replica server.

NIS_INVALIDOBJ

The object pointed to by *obj* is not a valid NIS+ object.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_LINKNAMEERROR

The name passed resolved to a *LINK* type object and the contents of the link pointed to an invalid name.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_MODFAIL

The attempted modification failed.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type `DIRECTORY`, which contains the type of namespace and contact information for a server within that namespace.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

ENVIRONMENT VARIABLES

NIS_PATH If the flag `EXPAND_NAME` is set, this variable is the search path used by `nis_lookup()`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_add()`, `nis_modify()`, and `nis_remove()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_server(3N)`, `nis_tables(3N)`

SunOS 5.7 Reference
Manual

`nis_error(3N)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name dirname, nis_server * machine); nis_error nis_rmdir(nis_name dirname, nis_server * machine); nis_error nis_servstate(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); nis_error nis_stats(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); void nis_freetags(nis_tag * tags, int numtags); nis_server ** nis_getservlist(nis_name dirname); void nis_freeservlist(nis_server ** machines);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><code>nis_mkdir()</code> is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i>, on a server, <i>machine</i>. If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_mkdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_rmdir()</code> is used to delete the directory, <i>dirname</i>, from the specified machine. The <i>machine</i> parameter cannot be <code>NULL</code>. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_rmdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_servstate()</code> is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, <code>nis_servstate()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>The <code>nis_stats()</code> function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <code>nisstat(1M)</code>.</p> <p><code>nis_servstate()</code> and <code>nis_stats()</code> use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <code><rpcsvc/nis_tags.h></code> which is included in <code><rpcsvc/nis.h></code>. Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to `NULL`. The *tag_value* pointers points to allocated string memory which contains the results. Use `nis_freetags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to `nis_objects(3N)`. `nis_freesservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_names(3N)`

SunOS 5.7 Reference
Manual

`nisstat(1M)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_frehtags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_rmdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_servstate(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); nis_error nis_stats(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); void nis_frehtags(nis_tag * <i>tags</i>, int <i>numtags</i>); nis_server ** nis_getservlist(nis_name <i>dirname</i>); void nis_freeservlist(nis_server ** <i>machines</i>);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p>nis_mkdir() is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i> , on a server, <i>machine</i> . If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <i>nis_server</i> structure, refer to <i>nis_objects(3N)</i> . To succeed, nis_mkdir() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>nis_rmdir() is used to delete the directory, <i>dirname</i> , from the specified machine. The <i>machine</i> parameter cannot be NULL . For a description of the <i>nis_server</i> structure, refer to <i>nis_objects(3N)</i> . To succeed, nis_rmdir() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>nis_servstate() is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, nis_servstate() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>The nis_stats() function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <i>nisstat(1M)</i> .</p> <p>nis_servstate() and nis_stats() use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <i><rpcsvc/nis_tags.h></i> which is included in <i><rpcsvc/nis.h></i> . Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to `NULL`. The *tag_value* pointers points to allocated string memory which contains the results. Use `nis_frehtags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to `nis_objects(3N)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_names(3N)`

SunOS 5.7 Reference
Manual

`nisstat(1M)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name dirname, nis_server * machine); nis_error nis_rmdir(nis_name dirname, nis_server * machine); nis_error nis_servstate(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); nis_error nis_stats(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); void nis_freetags(nis_tag * tags, int numtags); nis_server ** nis_getservlist(nis_name dirname); void nis_freeservlist(nis_server ** machines);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><code>nis_mkdir()</code> is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i>, on a server, <i>machine</i>. If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_mkdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_rmdir()</code> is used to delete the directory, <i>dirname</i>, from the specified machine. The <i>machine</i> parameter cannot be <code>NULL</code>. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_rmdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_servstate()</code> is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, <code>nis_servstate()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>The <code>nis_stats()</code> function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <code>nisstat(1M)</code>.</p> <p><code>nis_servstate()</code> and <code>nis_stats()</code> use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <code><rpcsvc/nis_tags.h></code> which is included in <code><rpcsvc/nis.h></code>. Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to `NULL`. The *tag_value* pointers points to allocated string memory which contains the results. Use `nis_freetags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to `nis_objects(3N)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_names(3N)`

SunOS 5.7 Reference
Manual

`nisstat(1M)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry - NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.`, with members `wickedwitch.west.oz.`

and `*.monkeys.west.oz.`, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.`, with members `toto.kansas`,

`@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_groups` man page refers to Solaris man pages only.

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
                 (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.` with members `wickedwitch.west.oz.` and `*.monkeys.west.oz.`, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.` with members `toto.kansas`, `@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

NOTES

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_groups` man page refers to Solaris man pages only.

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
    (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	<code>nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry</code> – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result *nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result *nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_first_entry(nis_name table_name); nis_result *nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[''] characters. Indexed names have the following form:</p> <pre>[colname=value, . . .],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.</p> <p>FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3N)</code>), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <i>name</i> does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.
NO_CACHE	This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.
MASTER_ONLY	This flag is even stronger than NO_CACHE in that it specifies that the client library should <i>only</i> get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling <code>nis_getnames()</code> [see <code>nis_local_names(3N)</code>] which uses the environment variable <code>NIS_PATH</code> .

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, *callback*, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See `NOTES`.

`nis_add_entry()` will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_add_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the *flags* variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table. To succeed, `nis_remove_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_modify_entry()` modifies an object identified by *name*. The parameter *object* should point to an entry with the `LEN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* : *zo_owner* , *zo_group* , and *zo_access* .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag MOD_SAMEOBJ then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the NIS_NOTSAMEOBJ error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag RETURN_RESULT has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, *nis_modify_entry()* must inherit the PAF_TRUSTED_PATH attribute.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the *nis_result* structure must be copied by the caller into local storage and passed as an argument to *nis_next_entry()* .

nis_next_entry() retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to *nis_next_entry()* there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error NIS_CHAINBROKEN is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type *nis_result* :

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` [see `nis_error(3N)`].

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` [see `nis_names(3N)`]. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` [see `nis_server(3N)`].

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE

The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST

A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED

The entry returned came from an object cache that has *expired*. This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_CBERROR

An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS

Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ

The object pointed to by *object* is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.

NIS_LINKNAMEERROR

The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL

The attempted modification failed for some reason.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.

NIS_NAMEUNREACHABLE

This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.

NIS_NOCALLBACK

The server was unable to contact the callback service on your machine. This results in no data being returned.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_NOT_ME

A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.

NIS_NOTFOUND

No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the `nis_remove()`.

If the `FOLLOW_PATH` flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.

NIS_NOTMASTER

A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the `/var/nis/NIS_SHARED_DIRCACHE` file will need to have their cache managers restarted (use `nis_cachemgr -i`) to flush this cache.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOTSEARCHABLE

The table name resolved to a NIS+ object that was not searchable.

NIS_PARTIAL

This result is similar to `NIS_NOTFOUND` except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

NIS_S_NOTFOUND

The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.

NIS_S_SUCCESS

Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.

NIS_SUCCESS

The request was successful.

NIS_SYSTEMERROR

Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.

NIS_TOOMANYATTRS

The search criteria passed to the server had more attributes than the table had searchable columns.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. `add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.

NIS_TYPEMISMATCH

An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

ENVIRONMENT VARIABLES

NIS_PATH When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

WARNINGS

NOTES

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3N)`, `nis_server(3N)`,
`rpc_svc_calls(3N)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3)`,
`nis_clone_object(3N)`, `nis_db(3N)`, `nis_objects(3N)`,
`nis_destroy_object(3N)`, `nis_error(3N)`, `nis_getnames(3N)`,
`nis_local_names(3N)`, `nis_objects(3N)`, `attributes(5)`

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling `nis_list()` with a callback from within a list callback function is not currently supported.

There are currently no known methods for `nis_first_entry()` and `nis_next_entry()` to get their answers from only the master server.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3N)`. Otherwise, this function is MT-Safe.

NAME	<code>nis_names</code> , <code>nis_lookup</code> , <code>nis_add</code> , <code>nis_remove</code> , <code>nis_modify</code> , <code>nis_freeresult</code> – NIS+ namespace functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result * nis_lookup(nis_name <i>name</i>, uint_t <i>flags</i>); nis_result * nis_add(nis_name <i>name</i>, nis_object * <i>obj</i>); nis_result * nis_remove(nis_name <i>name</i>, nis_object * <i>obj</i>); nis_result * nis_modify(nis_name <i>name</i>, nis_object * <i>obj</i>); void nis_freeresult(nis_result * <i>result</i>);</pre>
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3N)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3N)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in <code>nis_subr(3N)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with <code>nis_clone_object(3N)</code> (see <code>nis_subr(3N)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> takes two parameters, the name of the object to be resolved in <i>name</i>, and a <i>flags</i> parameter, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see <code>nis_tables(3N)</code>). The <code>nis_lookup()</code> function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically OR ing zero or more flags from the following list.</p>

FOLLOW_LINKS	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until the either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.
NO_CACHE	When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.
MASTER_ONLY	When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3N)</code>) which uses the environment variable <code>NIS_PATH</code> .

The status value may be translated to ascii text using the function `nis_sperrno()` [see `nis_error(3N)`].

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function `nis_add()` will take the object *obj* and add it to the NIS+ namespace with the name *name* . This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function `nis_remove()` will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not `NULL`, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the `NIS_NOTSAMEOBJ` error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function `nis_modify()` will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error `NIS_NOTSAMEOBJ` if the object identifier of the passed object does not match that of the object being modified in the namespace.

Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-null the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted `'.'`(dot) characters. If these conditions are not met the operation will fail and return the `NIS_BADNAME` error code.

Results

These functions return a pointer to a structure of type `nis_result`:

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` (see `nis_error(3N)`).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to `nis_freeresult()`. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` (see `nis_server(3N)`). Refer to `nis_objects(3N)` for a description of the *nis_object* structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

- zticks* The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
- dticks* The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
- aticks* The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
- cticks* The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

`NIS_SUCCESS`

The request was successful.

`NIS_S_SUCCESS`

The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag `NO_CACHE` when you call the lookup function.

`NIS_NOTFOUND`

The named object does not exist in the namespace.

`NIS_CACHEEXPIRED`

The object returned came from an object cache that has *expired* . The time to live value has gone to zero and the object may have changed. If the flag `NO_CACHE` was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

`NIS_NAMEUNREACHABLE`

A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the `HARD_LOOKUP` flag.

NIS_UNKNOWNOBJ

The object returned is of an unknown type.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. For the *add* , *remove* , and *modify* operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of `nis_list()` if the client specifies a callback and the server does not have enough resources to handle the callback.

NIS_SYSTEMERROR

A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.

NIS_NOT_ME

A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.

NIS_NOTMASTER

An attempt was made to update the database on a replica server.

NIS_INVALIDOBJ

The object pointed to by *obj* is not a valid NIS+ object.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_LINKNAMEERROR

The name passed resolved to a *LINK* type object and the contents of the link pointed to an invalid name.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_MODFAIL

The attempted modification failed.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY , which contains the type of namespace and contact information for a server within that namespace.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a syslog(3) message indicating why the RPC request failed.

ENVIRONMENT
VARIABLES

NIS_PATH If the flag EXPAND_NAME is set, this variable is the search path used by nis_lookup() .

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY
OF TRUSTED
SOLARIS
CHANGES

To succeed, nis_add() , nis_modify() , and nis_remove() must inherit the PAF_TRUSTED_PATH attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

nis_server(3N) , nis_tables(3N)

SunOS 5.7 Reference
Manual

nis_error(3N) , nis_objects(3N) , nis_subr(3N) , attributes(5)

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name dirname, nis_server * machine); nis_error nis_rmdir(nis_name dirname, nis_server * machine); nis_error nis_servstate(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); nis_error nis_stats(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); void nis_freetags(nis_tag * tags, int numtags); nis_server ** nis_getservlist(nis_name dirname); void nis_freeservlist(nis_server ** machines);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><code>nis_mkdir()</code> is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i> , on a server, <i>machine</i> . If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code> . To succeed, <code>nis_mkdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_rmdir()</code> is used to delete the directory, <i>dirname</i> , from the specified machine. The <i>machine</i> parameter cannot be <code>NULL</code> . For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code> . To succeed, <code>nis_rmdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_servstate()</code> is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, <code>nis_servstate()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>The <code>nis_stats()</code> function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <code>nisstat(1M)</code> .</p> <p><code>nis_servstate()</code> and <code>nis_stats()</code> use the tag list. This tag list is a variable length array of <code>nis_tag</code> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <code><rpcsvc/nis_tags.h></code> which is included in <code><rpcsvc/nis.h></code> . Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result* . If there is an error, **result* is set to NULL . The *tag_value* pointers points to allocated string memory which contains the results. Use *nis_freetags()* to free the tag structure.

nis_getservlist() returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname* . Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to *nis_objects(3N)* . *nis_freeservlist()* frees the list of servers returned by *nis_getservlist()* . Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY
OF TRUSTED
SOLARIS
CHANGES

To succeed, *nis_mkdir()* , *nis_rmdir()* , and *nis_servstat()* must inherit the PAF_TRUSTED_PATH attribute.

SEE ALSO
Trusted Solaris 7
Reference Manual

nis_names(3N)

SunOS 5.7 Reference
Manual

nisstat(1M) , *nis_objects(3N)* , *nis_subr(3N)* , *attributes(5)*

NAME	<code>nis_names</code> , <code>nis_lookup</code> , <code>nis_add</code> , <code>nis_remove</code> , <code>nis_modify</code> , <code>nis_freeresult</code> – NIS+ namespace functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result * nis_lookup(nis_name name, uint_t flags); nis_result * nis_add(nis_name name, nis_object * obj); nis_result * nis_remove(nis_name name, nis_object * obj); nis_result * nis_modify(nis_name name, nis_object * obj); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3N)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3N)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in <code>nis_subr(3N)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with <code>nis_clone_object(3N)</code> (see <code>nis_subr(3N)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> takes two parameters, the name of the object to be resolved in <i>name</i>, and a <i>flags</i> parameter, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see <code>nis_tables(3N)</code>). The <code>nis_lookup()</code> function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically OR ing zero or more flags from the following list.</p>

FOLLOW_LINKS	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until the either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.
NO_CACHE	When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.
MASTER_ONLY	When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3N)</code>) which uses the environment variable <code>NIS_PATH</code> .

The status value may be translated to ascii text using the function `nis_sperrno()` [see `nis_error(3N)`].

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function `nis_add()` will take the object *obj* and add it to the NIS+ namespace with the name *name* . This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function `nis_remove()` will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not `NULL`, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the `NIS_NOTSAMEOBJ` error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function `nis_modify()` will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error `NIS_NOTSAMEOBJ` if the object identifier of the passed object does not match that of the object being modified in the namespace.

Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-null the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted `'.'`(dot) characters. If these conditions are not met the operation will fail and return the `NIS_BADNAME` error code.

Results

These functions return a pointer to a structure of type `nis_result`:

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` (see `nis_error(3N)`).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to `nis_freeresult()`. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` (see `nis_server(3N)`). Refer to `nis_objects(3N)` for a description of the *nis_object* structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

- zticks* The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
- dticks* The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
- aticks* The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
- cticks* The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

`NIS_SUCCESS`

The request was successful.

`NIS_S_SUCCESS`

The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag `NO_CACHE` when you call the lookup function.

`NIS_NOTFOUND`

The named object does not exist in the namespace.

`NIS_CACHEEXPIRED`

The object returned came from an object cache that has *expired* . The time to live value has gone to zero and the object may have changed. If the flag `NO_CACHE` was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

`NIS_NAMEUNREACHABLE`

A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the `HARD_LOOKUP` flag.

NIS_UNKNOWNOBJ

The object returned is of an unknown type.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. For the *add* , *remove* , and *modify* operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of `nis_list()` if the client specifies a callback and the server does not have enough resources to handle the callback.

NIS_SYSTEMERROR

A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.

NIS_NOT_ME

A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.

NIS_NOTMASTER

An attempt was made to update the database on a replica server.

NIS_INVALIDOBJ

The object pointed to by *obj* is not a valid NIS+ object.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_LINKNAMEERROR

The name passed resolved to a *LINK* type object and the contents of the link pointed to an invalid name.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_MODFAIL

The attempted modification failed.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY , which contains the type of namespace and contact information for a server within that namespace.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a syslog(3) message indicating why the RPC request failed.

ENVIRONMENT
VARIABLES

NIS_PATH If the flag EXPAND_NAME is set, this variable is the search path used by nis_lookup() .

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY
OF TRUSTED
SOLARIS
CHANGES

To succeed, nis_add() , nis_modify() , and nis_remove() must inherit the PAF_TRUSTED_PATH attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

nis_server(3N) , nis_tables(3N)

SunOS 5.7 Reference
Manual

nis_error(3N) , nis_objects(3N) , nis_subr(3N) , attributes(5)

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.

NAME	<code>nis_tables</code> , <code>nis_list</code> , <code>nis_add_entry</code> , <code>nis_remove_entry</code> , <code>nis_modify_entry</code> , <code>nis_first_entry</code> , <code>nis_next_entry</code> – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result *nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result *nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_first_entry(nis_name table_name); nis_result *nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[''] characters. Indexed names have the following form:</p> <pre>[colname=value, . . .], tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.</p> <p>FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3N)</code>), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <i>name</i> does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.
NO_CACHE	This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.
MASTER_ONLY	This flag is even stronger than NO_CACHE in that it specifies that the client library should <i>only</i> get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling <code>nis_getnames()</code> [see <code>nis_local_names(3N)</code>] which uses the environment variable <code>NIS_PATH</code> .

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, *callback*, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See **NOTES**.

`nis_add_entry()` will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_add_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the *flags* variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table. To succeed, `nis_remove_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_modify_entry()` modifies an object identified by *name*. The parameter *object* should point to an entry with the `LEN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* : *zo_owner* , *zo_group* , and *zo_access* .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag `MOD_SAMEOBJ` then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the `NIS_NOTSAMEOBJ` error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_first_entry()` fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the `nis_result` structure must be copied by the caller into local storage and passed as an argument to `nis_next_entry()` .

`nis_next_entry()` retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to `nis_next_entry()` there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error `NIS_CHAINBROKEN` is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type `nis_result` :

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` [see `nis_error(3N)`].

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` [see `nis_names(3N)`]. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` [see `nis_server(3N)`].

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE

The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST

A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED

The entry returned came from an object cache that has *expired*. This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_CBERROR

An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS

Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ

The object pointed to by *object* is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.

NIS_LINKNAMEERROR

The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL

The attempted modification failed for some reason.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.

NIS_NAMEUNREACHABLE

This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.

NIS_NOCALLBACK

The server was unable to contact the callback service on your machine. This results in no data being returned.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_NOT_ME

A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.

NIS_NOTFOUND

No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the `nis_remove()`.

If the `FOLLOW_PATH` flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.

NIS_NOTMASTER

A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the `/var/nis/NIS_SHARED_DIRCACHE` file will need to have their cache managers restarted (use `nis_cachemgr -i`) to flush this cache.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOTSEARCHABLE

The table name resolved to a NIS+ object that was not searchable.

NIS_PARTIAL

This result is similar to `NIS_NOTFOUND` except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.

- NIS_RPCERROR
This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.
- NIS_S_NOTFOUND
The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.
- NIS_S_SUCCESS
Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.
- NIS_SUCCESS
The request was successful.
- NIS_SYSTEMERROR
Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.
- NIS_TOOMANYATTRS
The search criteria passed to the server had more attributes than the table had searchable columns.
- NIS_TRYAGAIN
The server connected to was too busy to handle your request.
`add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.
- NIS_TYPEMISMATCH
An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

ENVIRONMENT
VARIABLES

ATTRIBUTES

NIS_PATH When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`nis_cachemgr(1M)`, `nis_names(3N)`, `nis_server(3N)`,
`rpc_svc_calls(3N)`

**SunOS 5.7 Reference
Manual**

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3)`,
`nis_clone_object(3N)`, `nis_db(3N)`, `nis_objects(3N)`,
`nis_destroy_object(3N)`, `nis_error(3N)`, `nis_getnames(3N)`,
`nis_local_names(3N)`, `nis_objects(3N)`, `attributes(5)`

WARNINGS

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

NOTES

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling `nis_list()` with a callback from within a list callback function is not currently supported.

There are currently no known methods for `nis_first_entry()` and `nis_next_entry()` to get their answers from only the master server.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3N)`. Otherwise, this function is MT-Safe.

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]</pre> <pre>#include <rpcsvc/nis.h> nis_result *nis_lookup(nis_name name, uint_t flags); nis_result *nis_add(nis_name name, nis_object *obj); nis_result *nis_remove(nis_name name, nis_object *obj); nis_result *nis_modify(nis_name name, nis_object *obj); void nis_freeresult(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3N)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3N)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in <code>nis_subr(3N)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with <code>nis_clone_object(3N)</code> (see <code>nis_subr(3N)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> takes two parameters, the name of the object to be resolved in <i>name</i>, and a flags parameter, <i>flags</i>, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see <code>nis_tables(3N)</code>). The <code>nis_lookup()</code> function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically OR ing zero or more flags from the following list.</p>

FOLLOW_LINKS	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.
NO_CACHE	When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.
MASTER_ONLY	When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3N)</code>) which uses the environment variable <code>NIS_PATH</code> .

The status value may be translated to ascii text using the function `nis_sperrno()` [see `nis_error(3N)`].

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the FOLLOW_LINKS flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function `nis_add()` will take the object *obj* and add it to the NIS+ namespace with the name *name* . This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function `nis_remove()` will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not `NULL`, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the `NIS_NOTSAMEOBJ` error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function `nis_modify()` will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error `NIS_NOTSAMEOBJ` if the object identifier of the passed object does not match that of the object being modified in the namespace.

Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-null the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted '.'(dot) characters. If these conditions are not met the operation will fail and return the `NIS_BADNAME` error code.

Results

These functions return a pointer to a structure of type `nis_result`:

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` (see `nis_error(3N)`).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to `nis_freeresult()`. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` (see `nis_server(3N)`). Refer to `nis_objects(3N)` for a description of the *nis_object* structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

- zticks* The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
- dticks* The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
- aticks* The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
- cticks* The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

`NIS_SUCCESS`

The request was successful.

`NIS_S_SUCCESS`

The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag `NO_CACHE` when you call the lookup function.

`NIS_NOTFOUND`

The named object does not exist in the namespace.

`NIS_CACHEEXPIRED`

The object returned came from an object cache that has *expired* . The time to live value has gone to zero and the object may have changed. If the flag `NO_CACHE` was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

`NIS_NAMEUNREACHABLE`

A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the `HARD_LOOKUP` flag.

NIS_UNKNOWNOBJ

The object returned is of an unknown type.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. For the *add*, *remove*, and *modify* operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of `nis_list()` if the client specifies a callback and the server does not have enough resources to handle the callback.

NIS_SYSTEMERROR

A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.

NIS_NOT_ME

A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.

NIS_NOTMASTER

An attempt was made to update the database on a replica server.

NIS_INVALIDOBJ

The object pointed to by *obj* is not a valid NIS+ object.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_LINKNAMEERROR

The name passed resolved to a *LINK* type object and the contents of the link pointed to an invalid name.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_MODFAIL

The attempted modification failed.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type `DIRECTORY`, which contains the type of namespace and contact information for a server within that namespace.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

ENVIRONMENT VARIABLES

NIS_PATH If the flag `EXPAND_NAME` is set, this variable is the search path used by `nis_lookup()`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_add()`, `nis_modify()`, and `nis_remove()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_server(3N)`, `nis_tables(3N)`

SunOS 5.7 Reference
Manual

`nis_error(3N)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.

NAME	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result *nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result *nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_first_entry(nis_name table_name); nis_result *nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[''] characters. Indexed names have the following form:</p> <pre>[colname=value, . . .],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.</p> <p>FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3N)</code>), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <i>name</i> does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.
NO_CACHE	This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.
MASTER_ONLY	This flag is even stronger than NO_CACHE in that it specifies that the client library should <i>only</i> get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling <code>nis_getnames()</code> [see <code>nis_local_names(3N)</code>] which uses the environment variable <code>NIS_PATH</code> .

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, *callback*, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See **NOTES**.

`nis_add_entry()` will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_add_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the *flags* variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table. To succeed, `nis_remove_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_modify_entry()` modifies an object identified by *name*. The parameter *object* should point to an entry with the `LEN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* : *zo_owner* , *zo_group* , and *zo_access* .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag MOD_SAMEOBJ then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the NIS_NOTSAMEOBJ error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag RETURN_RESULT has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, *nis_modify_entry()* must inherit the PAF_TRUSTED_PATH attribute.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the *nis_result* structure must be copied by the caller into local storage and passed as an argument to *nis_next_entry()* .

nis_next_entry() retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to *nis_next_entry()* there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error NIS_CHAINBROKEN is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type *nis_result* :

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` [see `nis_error(3N)`].

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` [see `nis_names(3N)`]. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` [see `nis_server(3N)`].

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE

The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST

A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED

The entry returned came from an object cache that has *expired*. This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_CBERROR

An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS

Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ

The object pointed to by *object* is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.

NIS_LINKNAMEERROR

The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL

The attempted modification failed for some reason.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.

NIS_NAMEUNREACHABLE

This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.

NIS_NOCALLBACK

The server was unable to contact the callback service on your machine. This results in no data being returned.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_NOT_ME

A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.

NIS_NOTFOUND

No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the `nis_remove()`.

If the `FOLLOW_PATH` flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.

NIS_NOTMASTER

A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the `/var/nis/NIS_SHARED_DIRCACHE` file will need to have their cache managers restarted (use `nis_cachemgr -i`) to flush this cache.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOTSEARCHABLE

The table name resolved to a NIS+ object that was not searchable.

NIS_PARTIAL

This result is similar to `NIS_NOTFOUND` except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

NIS_S_NOTFOUND

The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.

NIS_S_SUCCESS

Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.

NIS_SUCCESS

The request was successful.

NIS_SYSTEMERROR

Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.

NIS_TOOMANYATTRS

The search criteria passed to the server had more attributes than the table had searchable columns.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. `add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.

NIS_TYPEMISMATCH

An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

ENVIRONMENT VARIABLES

NIS_PATH When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

WARNINGS

NOTES

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3N)`, `nis_server(3N)`,
`rpc_svc_calls(3N)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3)`,
`nis_clone_object(3N)`, `nis_db(3N)`, `nis_objects(3N)`,
`nis_destroy_object(3N)`, `nis_error(3N)`, `nis_getnames(3N)`,
`nis_local_names(3N)`, `nis_objects(3N)`, `attributes(5)`

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling `nis_list()` with a callback from within a list callback function is not currently supported.

There are currently no known methods for `nis_first_entry()` and `nis_next_entry()` to get their answers from only the master server.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3N)`. Otherwise, this function is MT-Safe.

NAME	nis_ping, nis_checkpoint – Misc NIS+ log administration functions				
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]</pre> <pre>#include <rpcsvc/nis.h></pre> <pre>void nis_ping(nis_name <i>dirname</i>, uint32_t <i>utime</i>, nis_object * <i>dirobj</i>);</pre> <pre>nis_result * nis_checkpoint(nis_name <i>dirname</i>);</pre>				
DESCRIPTION	<p>nis_ping() is called by the master server for a directory when a change has occurred within that directory. The parameter <i>dirname</i> identifies the directory with the change. If the parameter <i>dirobj</i> is <code>NULL</code>, this function looks up the directory object for <i>dirname</i> and uses the list of replicas it contains. The parameter <i>utime</i> contains the timestamp of the last change made to the directory. This timestamp is used by the replicas when retrieving updates made to the directory.</p> <p>The effect of calling nis_ping() is to schedule an update on the replica. A short time after a ping is received, typically about two minutes, the replica compares the last update time for its databases to the timestamp sent by the ping. If the ping timestamp is later, the replica establishes a connection with the master server and request all changes from the log that occurred after the last update that it had recorded in its local log.</p> <p>To succeed, nis_ping() must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>nis_checkpoint() is used to force the service to checkpoint information that has been entered in the log but has not been checkpointed to disk. When called, this function checkpoints the database for each table in the directory, the database containing the directory and the transaction log. Care should be used in calling this function since directories that have seen a lot of changes may take several minutes to checkpoint. During the checkpointing process, the service will be unavailable for updates for all directories that are served by this machine as master.</p> <p>nis_checkpoint() returns a pointer to a <i>nis_result</i> structure (described in nis_tables(3N)). This structure should be freed with nis_freeresult() (see nis_names(3N)). The only items of interest in the returned result are the status value and the statistics.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <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>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	MT-Safe				

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual**

To succeed, `nis_ping()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_names(3N)` , `nis_tables(3N)`

`nislog(1M)` , `nisfiles(4)` , `attributes(5)`

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error_t nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error_t nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error_t nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error_t nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error_t nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.` with members `wickedwitch.west.oz.` and `*.monkeys.west.oz.`, the function call

```
bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");
```

will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.` with members `toto.kansas`, `@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

NOTES

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_groups` man page refers to Solaris man pages only.

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
    (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	nis_names, nis_lookup, nis_add, nis_remove, nis_modify, nis_freeresult – NIS+ namespace functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...]</pre> <pre>#include <rpcsvc/nis.h> nis_result *nis_lookup(nis_name name, uint_t flags); nis_result *nis_add(nis_name name, nis_object *obj); nis_result *nis_remove(nis_name name, nis_object *obj); nis_result *nis_modify(nis_name name, nis_object *obj); void nis_freeresult(nis_result *result);</pre>
DESCRIPTION	<p>These functions are used to locate and manipulate all NIS+ objects (see <code>nis_objects(3N)</code>) except the NIS+ entry objects. To look up the NIS+ entry objects within a NIS+ table, refer to <code>nis_subr(3N)</code>.</p> <p><code>nis_lookup()</code> resolves a NIS+ name and returns a copy of that object from a NIS+ server. <code>nis_add()</code> and <code>nis_remove()</code> add and remove objects to the NIS+ namespace, respectively. <code>nis_modify()</code> can change specific attributes of an object that already exists in the namespace.</p> <p>These functions should be used only with names that refer to an NIS+ Directory, NIS+ Table, NIS+ Group, or NIS+ Private object. If a name refers to an NIS+ entry object, the functions listed in <code>nis_subr(3N)</code> should be used.</p> <p><code>nis_freeresult()</code> frees all memory associated with a <code>nis_result</code> structure. This function must be called to free the memory associated with a NIS+ result. <code>nis_lookup()</code>, <code>nis_add()</code>, <code>nis_remove()</code>, and <code>nis_modify()</code> all return a pointer to a <code>nis_result</code> structure which <i>must</i> be freed by calling <code>nis_freeresult()</code> when you have finished using it. If one or more of the objects returned in the structure need to be retained, they can be copied with <code>nis_clone_object(3N)</code> (see <code>nis_subr(3N)</code>). To succeed, <code>nis_add()</code>, <code>nis_modify()</code>, and <code>nis_remove()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_lookup()</code> takes two parameters, the name of the object to be resolved in <i>name</i>, and a flags parameter, <i>flags</i>, which is defined below. The object name is expected to correspond to the syntax of a non-indexed NIS+ name (see <code>nis_tables(3N)</code>). The <code>nis_lookup()</code> function is the <i>only</i> function from this group that can use a non-fully qualified name. If the parameter <i>name</i> is not a fully qualified name, then the flag <code>EXPAND_NAME</code> <i>must</i> be specified in the call. If this flag is not specified, the function will fail with the error <code>NIS_BADNAME</code>.</p> <p>The <i>flags</i> parameter is constructed by logically OR ing zero or more flags from the following list.</p>

FOLLOW_LINKS	When specified, the client library will “follow” links by issuing another NIS+ lookup call for the object named by the link. If the linked object is itself a link, then this process will iterate until either a object is found that is not a <i>LINK</i> type object, or the library has followed 16 links.
HARD_LOOKUP	When specified, the client library will retry the lookup until it is answered by a server. Using this flag will cause the library to block until at least one NIS+ server is available. If the network connectivity is impaired, this can be a relatively long time.
NO_CACHE	When specified, the client library will bypass any object caches and will get the object from either the master NIS+ server or one of its replicas.
MASTER_ONLY	When specified, the client library will bypass any object caches and any domain replicas and fetch the object from the NIS+ master server for the object’s domain. This insures that the object returned is up to date at the cost of a possible performance degradation and failure if the master server is unavailable or physically distant.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling the function <code>nis_getnames()</code> (see <code>nis_subr(3N)</code>) which uses the environment variable <code>NIS_PATH</code> .

The status value may be translated to ascii text using the function `nis_sperrno()` [see `nis_error(3N)`].

On return, the *objects* array in the result will contain one and possibly several objects that were resolved by the request. If the `FOLLOW_LINKS` flag was present, on success the function could return several entry objects if the link in question pointed within a table. If an error occurred when following a link, the objects array will contain a copy of the link object itself.

The function `nis_add()` will take the object *obj* and add it to the NIS+ namespace with the name *name* . This operation will fail if the client making the request does not have the *create* access right for the domain in which this object will be added. The parameter *name* must contain a fully qualified NIS+ name. The object members *zo_name* and *zo_domain* will be constructed from this name. This operation will fail if the object already exists. This feature prevents the accidental addition of objects over another object that has been added by another process.

The function `nis_remove()` will remove the object with name *name* from the NIS+ namespace. The client making this request must have the *destroy* access right for the domain in which this object resides. If the named object is a link, the link is removed and *not* the object that it points to. If the parameter *obj* is not `NULL`, it is assumed to point to a copy of the object being removed. In this case, if the object on the server does not have the same object identifier as the object being passed, the operation will fail with the `NIS_NOTSAMEOBJ` error. This feature allows the client to insure that it is removing the desired object. The parameter *name* must contain a fully qualified NIS+ name.

The function `nis_modify()` will modify the object named by *name* to the field values in the object pointed to by *obj*. This object should contain a copy of the object from the name space that is being modified. This operation will fail with the error `NIS_NOTSAMEOBJ` if the object identifier of the passed object does not match that of the object being modified in the namespace.

Normally the contents of the member *zo_name* in the *nis_object* structure would be constructed from the name passed in the *name* parameter. However, if it is non-null the client library will use the name in the *zo_name* member to perform a rename operation on the object. This name *must not* contain any unquoted '.'(dot) characters. If these conditions are not met the operation will fail and return the `NIS_BADNAME` error code.

Results

These functions return a pointer to a structure of type `nis_result`:

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` (see `nis_error(3N)`).

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by the call to `nis_freeresult()`. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` (see `nis_server(3N)`). Refer to `nis_objects(3N)` for a description of the *nis_object* structure.

The various ticks contain details of where the time was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

- zticks* The time spent in the NIS+ service itself. This count starts when the server receives the request and stops when it sends the reply.
- dticks* The time spent in the database backend. This time is measured from the time a database call starts, until the result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.
- aticks* The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.
- cticks* The total time spent in the request. This clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value, you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

RETURN VALUES

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

`NIS_SUCCESS`

The request was successful.

`NIS_S_SUCCESS`

The request was successful, however the object returned came from an object cache and not directly from the server. If you do not wish to see objects from object caches you must specify the flag `NO_CACHE` when you call the lookup function.

`NIS_NOTFOUND`

The named object does not exist in the namespace.

`NIS_CACHEEXPIRED`

The object returned came from an object cache that has *expired* . The time to live value has gone to zero and the object may have changed. If the flag `NO_CACHE` was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

`NIS_NAMEUNREACHABLE`

A server for the directory of the named object could not be reached. This can occur when there is a network partition or all servers have crashed. See the `HARD_LOOKUP` flag.

NIS_UNKNOWNOBJ

The object returned is of an unknown type.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. For the *add* , *remove* , and *modify* operations this is returned when either the master server for a directory is unavailable or it is in the process of checkpointing its database. It can also be returned when the server is updating its internal state. And in the case of `nis_list()` if the client specifies a callback and the server does not have enough resources to handle the callback.

NIS_SYSTEMERROR

A generic system error occurred while attempting the request. Most commonly the server has crashed or the database has become corrupted. Check the syslog record for error messages from the server.

NIS_NOT_ME

A request was made to a server that does not serve the name in question. Normally this will not occur, however if you are not using the built in location mechanism for servers you may see this if your mechanism is broken.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new object or modify the existing named object.

NIS_NOTMASTER

An attempt was made to update the database on a replica server.

NIS_INVALIDOBJ

The object pointed to by *obj* is not a valid NIS+ object.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_LINKNAMEERROR

The name passed resolved to a *LINK* type object and the contents of the link pointed to an invalid name.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_MODFAIL

The attempted modification failed.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type `DIRECTORY`, which contains the type of namespace and contact information for a server within that namespace.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

**ENVIRONMENT
VARIABLES**

NIS_PATH If the flag `EXPAND_NAME` is set, this variable is the search path used by `nis_lookup()`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, `nis_add()`, `nis_modify()`, and `nis_remove()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_server(3N)`, `nis_tables(3N)`

SunOS 5.7 Reference
Manual

`nis_error(3N)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NOTES

You cannot modify the name of an object if that modification would cause the object to reside in a different domain.

You cannot modify the schema of a table object.

NAME	nis_tables, nis_list, nis_add_entry, nis_remove_entry, nis_modify_entry, nis_first_entry, nis_next_entry – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result *nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result *nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_first_entry(nis_name table_name); nis_result *nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[]' characters. Indexed names have the following form:</p> <pre>[colname=value, . . .],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.</p> <p>FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3N)</code>), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <i>name</i> does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.
NO_CACHE	This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.
MASTER_ONLY	This flag is even stronger than NO_CACHE in that it specifies that the client library should <i>only</i> get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling <code>nis_getnames()</code> [see <code>nis_local_names(3N)</code>] which uses the environment variable <code>NIS_PATH</code> .

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, *callback*, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See **NOTES**.

`nis_add_entry()` will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_add_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the *flags* variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table. To succeed, `nis_remove_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_modify_entry()` modifies an object identified by *name*. The parameter *object* should point to an entry with the `LEN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* : *zo_owner* , *zo_group* , and *zo_access* .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag MOD_SAMEOBJ then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the NIS_NOTSAMEOBJ error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag RETURN_RESULT has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, *nis_modify_entry()* must inherit the PAF_TRUSTED_PATH attribute.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the *nis_result* structure must be copied by the caller into local storage and passed as an argument to *nis_next_entry()* .

nis_next_entry() retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to *nis_next_entry()* there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error NIS_CHAINBROKEN is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type *nis_result* :

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` [see `nis_error(3N)`].

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` [see `nis_names(3N)`]. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` [see `nis_server(3N)`].

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE

The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST

A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED

The entry returned came from an object cache that has *expired*. This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_CBERROR

An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS

Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ

The object pointed to by *object* is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.

NIS_LINKNAMEERROR

The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL

The attempted modification failed for some reason.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.

NIS_NAMEUNREACHABLE

This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.

NIS_NOCALLBACK

The server was unable to contact the callback service on your machine. This results in no data being returned.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_NOT_ME

A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.

NIS_NOTFOUND

No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the `nis_remove()`.

If the `FOLLOW_PATH` flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.

NIS_NOTMASTER

A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the `/var/nis/NIS_SHARED_DIRCACHE` file will need to have their cache managers restarted (use `nis_cachemgr -i`) to flush this cache.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOTSEARCHABLE

The table name resolved to a NIS+ object that was not searchable.

NIS_PARTIAL

This result is similar to `NIS_NOTFOUND` except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

NIS_S_NOTFOUND

The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.

NIS_S_SUCCESS

Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.

NIS_SUCCESS

The request was successful.

NIS_SYSTEMERROR

Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.

NIS_TOOMANYATTRS

The search criteria passed to the server had more attributes than the table had searchable columns.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. `add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.

NIS_TYPEMISMATCH

An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

ENVIRONMENT VARIABLES

NIS_PATH When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

WARNINGS

NOTES

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3N)`, `nis_server(3N)`,
`rpc_svc_calls(3N)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3)`,
`nis_clone_object(3N)`, `nis_db(3N)`, `nis_objects(3N)`,
`nis_destroy_object(3N)`, `nis_error(3N)`, `nis_getnames(3N)`,
`nis_local_names(3N)`, `nis_objects(3N)`, `attributes(5)`

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling `nis_list()` with a callback from within a list callback function is not currently supported.

There are currently no known methods for `nis_first_entry()` and `nis_next_entry()` to get their answers from only the master server.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3N)`. Otherwise, this function is MT-Safe.

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.` with members `wickedwitch.west.oz.` and `*.monkeys.west.oz.`, the function call

`bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");` will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.` with members `toto.kansas`, `@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

NOTES

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_groups` man page refers to Solaris man pages only.

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
    (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_rmdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_servstate(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); nis_error nis_stats(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); void nis_freetags(nis_tag * <i>tags</i>, int <i>numtags</i>); nis_server ** nis_getservlist(nis_name <i>dirname</i>); void nis_freeservlist(nis_server ** <i>machines</i>);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p>nis_mkdir() is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i> , on a server, <i>machine</i> . If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <i>nis_server</i> structure, refer to <i>nis_objects(3N)</i> . To succeed, nis_mkdir() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>nis_rmdir() is used to delete the directory, <i>dirname</i> , from the specified machine. The <i>machine</i> parameter cannot be NULL . For a description of the <i>nis_server</i> structure, refer to <i>nis_objects(3N)</i> . To succeed, nis_rmdir() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>nis_servstate() is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, nis_servstate() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>The nis_stats() function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <i>nisstat(1M)</i> .</p> <p>nis_servstate() and nis_stats() use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <i><rpcsvc/nis_tags.h></i> which is included in <i><rpcsvc/nis.h></i> . Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to `NULL`. The *tag_value* pointers points to allocated string memory which contains the results. Use `nis_freetags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to `nis_objects(3N)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_names(3N)`

SunOS 5.7 Reference
Manual

`nisstat(1M)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name dirname, nis_server * machine); nis_error nis_rmdir(nis_name dirname, nis_server * machine); nis_error nis_servstate(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); nis_error nis_stats(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); void nis_freetags(nis_tag * tags, int numtags); nis_server ** nis_getservlist(nis_name dirname); void nis_freeservlist(nis_server ** machines);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><code>nis_mkdir()</code> is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i>, on a server, <i>machine</i>. If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_mkdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_rmdir()</code> is used to delete the directory, <i>dirname</i>, from the specified machine. The <i>machine</i> parameter cannot be <code>NULL</code>. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_rmdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_servstate()</code> is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, <code>nis_servstate()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>The <code>nis_stats()</code> function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <code>nisstat(1M)</code>.</p> <p><code>nis_servstate()</code> and <code>nis_stats()</code> use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <code><rpcsvc/nis_tags.h></code> which is included in <code><rpcsvc/nis.h></code>. Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to `NULL`. The *tag_value* pointers points to allocated string memory which contains the results. Use `nis_freetags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to `nis_objects(3N)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_names(3N)`

SunOS 5.7 Reference
Manual

`nisstat(1M)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name dirname, nis_server * machine); nis_error nis_rmdir(nis_name dirname, nis_server * machine); nis_error nis_servstate(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); nis_error nis_stats(nis_server * machine, nis_tag * tags, int numtags, nis_tag ** result); void nis_freetags(nis_tag * tags, int numtags); nis_server ** nis_getservlist(nis_name dirname); void nis_freeservlist(nis_server ** machines);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p><code>nis_mkdir()</code> is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i>, on a server, <i>machine</i>. If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_mkdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_rmdir()</code> is used to delete the directory, <i>dirname</i>, from the specified machine. The <i>machine</i> parameter cannot be <code>NULL</code>. For a description of the <code>nis_server</code> structure, refer to <code>nis_objects(3N)</code>. To succeed, <code>nis_rmdir()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p><code>nis_servstate()</code> is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, <code>nis_servstate()</code> must inherit the <code>PAF_TRUSTED_PATH</code> attribute.</p> <p>The <code>nis_stats()</code> function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <code>nisstat(1M)</code>.</p> <p><code>nis_servstate()</code> and <code>nis_stats()</code> use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <code><rpcsvc/nis_tags.h></code> which is included in <code><rpcsvc/nis.h></code>. Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to `NULL`. The *tag_value* pointers points to allocated string memory which contains the results. Use `nis_freetags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to `nis_objects(3N)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_names(3N)`

SunOS 5.7 Reference
Manual

`nisstat(1M)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME	nis_server, nis_mkdir, nis_rmdir, nis_servstate, nis_stats, nis_getservlist, nis_freeservlist, nis_freetags – Miscellaneous NIS+ functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_error nis_mkdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_rmdir(nis_name <i>dirname</i>, nis_server * <i>machine</i>); nis_error nis_servstate(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); nis_error nis_stats(nis_server * <i>machine</i>, nis_tag * <i>tags</i>, int <i>numtags</i>, nis_tag ** <i>result</i>); void nis_freetags(nis_tag * <i>tags</i>, int <i>numtags</i>); nis_server ** nis_getservlist(nis_name <i>dirname</i>); void nis_freeservlist(nis_server ** <i>machines</i>);</pre>
DESCRIPTION	<p>These functions provide a variety of services for NIS+ applications.</p> <p>nis_mkdir() is used to create the necessary databases to support NIS+ service for a directory, <i>dirname</i> , on a server, <i>machine</i> . If this operation is successful, it means that the directory object describing <i>dirname</i> has been updated to reflect that server <i>machine</i> is serving the named directory. For a description of the <i>nis_server</i> structure, refer to <i>nis_objects(3N)</i> . To succeed, nis_mkdir() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>nis_rmdir() is used to delete the directory, <i>dirname</i> , from the specified machine. The <i>machine</i> parameter cannot be NULL . For a description of the <i>nis_server</i> structure, refer to <i>nis_objects(3N)</i> . To succeed, nis_rmdir() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>nis_servstate() is used to set and read the various state variables of the NIS+ servers. In particular the internal debugging state of the servers may be set and queried. To succeed, nis_servstate() must inherit the PAF_TRUSTED_PATH attribute.</p> <p>The nis_stats() function is used to retrieve statistics about how the server is operating. Tracking these statistics can help administrators determine when they need to add additional replicas or to break up a domain into two or more subdomains. For more information on reading statistics, see <i>nisstat(1M)</i> .</p> <p>nis_servstate() and nis_stats() use the tag list. This tag list is a variable length array of <i>nis_tag</i> structures whose length is passed to the function in the <i>numtags</i> parameter. The set of legal tags are defined in the file <i><rpcsvc/nis_tags.h></i> which is included in <i><rpcsvc/nis.h></i> . Because these tags can and do vary between implementations of the NIS+ service, it is</p>

best to consult this file for the supported list. Passing unrecognized tags to a server will result in their *tag_value* member being set to the string “unknown.” Both of these functions return their results in malloced tag structure, **result*. If there is an error, **result* is set to `NULL`. The *tag_value* pointers points to allocated string memory which contains the results. Use `nis_freetags()` to free the tag structure.

`nis_getservlist()` returns a null terminated list of *nis_server* structures that represent the list of servers that serve the domain named *dirname*. Servers from this list can be used when calling functions that require the name of a NIS+ server. For a description of the *nis_server* structure, refer to `nis_objects(3N)`. `nis_freeservlist()` frees the list of servers returned by `nis_getservlist()`. Note that this is the only legal way to free that list.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

To succeed, `nis_mkdir()`, `nis_rmdir()`, and `nis_servstat()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

`nis_names(3N)`

SunOS 5.7 Reference
Manual

`nisstat(1M)`, `nis_objects(3N)`, `nis_subr(3N)`, `attributes(5)`

NAME	<code>nis_tables</code> , <code>nis_list</code> , <code>nis_add_entry</code> , <code>nis_remove_entry</code> , <code>nis_modify_entry</code> , <code>nis_first_entry</code> , <code>nis_next_entry</code> – NIS+ table functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> nis_result *nis_list(nis_name name, uint_t flags, int (*callback)(nis_name table_name, nis_object * object, void * userdata), void * userdata); nis_result *nis_add_entry(nis_name table_name, nis_object * object, uint_t flags); nis_result *nis_remove_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_modify_entry(nis_name name, nis_object * object, uint_t flags); nis_result *nis_first_entry(nis_name table_name); nis_result *nis_next_entry(nis_name table_name, netobj * cookie); void nis_freeresult(nis_result * result);</pre>
DESCRIPTION	<p>These functions are used to search and modify NIS+ tables. <code>nis_list()</code> is used to search a table in the NIS+ namespace. <code>nis_first_entry()</code> and <code>nis_next_entry()</code> are used to enumerate a table one entry at a time. <code>nis_add_entry()</code>, <code>nis_remove_entry()</code>, and <code>nis_modify_entry()</code> are used to change the information stored in a table. <code>nis_freeresult()</code> is used to free the memory associated with the <code>nis_result</code> structure.</p> <p>Entries within a table are named by NIS+ indexed names. An indexed name is a compound name that is composed of a search criteria and a simple NIS+ name that identifies a table object. A search criteria is a series of column names and their associated values enclosed in bracket '[''] characters. Indexed names have the following form:</p> <pre>[colname=value, . . .],tablename</pre> <p>The list function, <code>nis_list()</code>, takes an indexed name as the value for the <i>name</i> parameter. Here, the <i>tablename</i> should be a fully qualified NIS+ name unless the <code>EXPAND_NAME</code> flag (described below) is set. The second parameter, <i>flags</i>, defines how the function will respond to various conditions. The value for this parameter is created by logically OR ing together one or more flags from the following list.</p> <p>FOLLOW_LINKS If the table specified in <i>name</i> resolves to be a <code>LINK</code> type object (see <code>nis_objects(3N)</code>), this flag specifies that the client library follow that link and do the search at that object. If this flag is not set and the name resolves to a link, the error <code>NIS_NOTSEARCHABLE</code> will be returned.</p>

FOLLOW_PATH	This flag specifies that if the entry is not found within this table, the list operation should follow the path specified in the table object. When used in conjunction with the ALL_RESULTS flag below, it specifies that the path should be followed regardless of the result of the search. When used in conjunction with the FOLLOW_LINKS flag above, named tables in the path that resolve to links will be followed until the table they point to is located. If a table in the path is not reachable because no server that serves it is available, the result of the operation will be either a “soft” success or a “soft” failure to indicate that not all tables in the path could be searched. If a name in the path names is either an invalid or non-existent object then it is silently ignored.
HARD_LOOKUP	This flag specifies that the operation should continue trying to contact a server of the named table until a definitive result is returned (such as NIS_NOTFOUND).
ALL_RESULTS	This flag can only be used in conjunction with FOLLOW_PATH and a callback function. When specified, it forces all of the tables in the path to be searched. If <i>name</i> does not specify a search criteria (imply that all entries are to be returned), then this flag will cause all of the entries in all of the tables in the path to be returned.
NO_CACHE	This flag specifies that the client library should bypass any client object caches and get its information directly from either the master server or a replica server for the named table.
MASTER_ONLY	This flag is even stronger than NO_CACHE in that it specifies that the client library should <i>only</i> get its information from the master server for a particular table. This guarantees that the information will be up to date. However, there may be severe performance penalties associated with contacting the master server directly on large networks. When used in conjunction with the HARD_LOOKUP flag, this will block the list operation until the master server is up and available.
EXPAND_NAME	When specified, the client library will attempt to expand a partially qualified name by calling <code>nis_getnames()</code> [see <code>nis_local_names(3N)</code>] which uses the environment variable <code>NIS_PATH</code> .

RETURN_RESULT This flag is used to specify that a copy of the returning object be returned in the `nis_result` structure if the operation was successful.

The third parameter to `nis_list()`, *callback*, is an optional pointer to a function that will process the `ENTRY` type objects that are returned from the search. If this pointer is `NULL`, then all entries that match the search criteria are returned in the `nis_result` structure, otherwise this function will be called once for each entry returned. When called, this function should return 0 when additional objects are desired and 1 when it no longer wishes to see any more objects. The fourth parameter, *userdata*, is simply passed to callback function along with the returned entry object. The client can use this pointer to pass state information or other relevant data that the callback function might need to process the entries.

The `nis_list()` function is not MT-Safe with callbacks. See **NOTES**.

`nis_add_entry()` will add the NIS+ object to the NIS+ *table_name*. The *flags* parameter is used to specify the failure semantics for the add operation. The default (*flags* equal 0) is to fail if the entry being added already exists in the table. The `ADD_OVERWRITE` flag may be used to specify that existing object is to be overwritten if it exists, (a modify operation) or added if it does not exist. With the `ADD_OVERWRITE` flag, this function will fail with the error `NIS_PERMISSION` if the existing object does not allow modify privileges to the client.

If the flag `RETURN_RESULT` has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, `nis_add_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_remove_entry()` removes the identified entry from the table or a set of entries identified by *table_name*. If the parameter *object* is non-null, it is presumed to point to a cached copy of the entry. When the removal is attempted, and the object that would be removed is not the same as the cached object pointed to by *object* then the operation will fail with an `NIS_NOTSAMEOBJ` error. If an object is passed with this function, the search criteria in *name* is optional as it can be constructed from the values within the entry. However, if no object is present, the search criteria must be included in the *name* parameter. If the *flags* variable is null, and the search criteria does not uniquely identify an entry, the `NIS_NOTUNIQUE` error is returned and the operation is aborted. If the flag parameter `REM_MULTIPLE` is passed, and if remove permission is allowed for each of these objects, then all objects that match the search criteria will be removed. Note that a null search criteria and the `REM_MULTIPLE` flag will remove all entries in a table. To succeed, `nis_remove_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_modify_entry()` modifies an object identified by *name*. The parameter *object* should point to an entry with the `LEN_MODIFIED` flag set in each column that contains new information.

The owner, group, and access rights of an entry are modified by placing the modified information into the respective fields of the parameter, *object* : *zo_owner* , *zo_group* , and *zo_access* .

These columns will replace their counterparts in the entry that is stored in the table. The entry passed must have the same number of columns, same type, and valid data in the modified columns for this operation to succeed.

If the flags parameter contains the flag MOD_SAMEOBJ then the object pointed to by *object* is assumed to be a cached copy of the original object. If the OID of the object passed is different than the OID of the object the server fetches, then the operation fails with the NIS_NOTSAMEOBJ error. This can be used to implement a simple read-modify-write protocol which will fail if the object is modified before the client can write the object back.

If the flag RETURN_RESULT has been specified, the server will return a copy of the resulting object if the operation was successful. To succeed, *nis_modify_entry()* must inherit the PAF_TRUSTED_PATH attribute.

nis_first_entry() fetches entries from a table one at a time. This mode of operation is extremely inefficient and callbacks should be used instead wherever possible. The table containing the entries of interest is identified by *name* . If a search criteria is present in *name* it is ignored. The value of *cookie* within the *nis_result* structure must be copied by the caller into local storage and passed as an argument to *nis_next_entry()* .

nis_next_entry() retrieves the “next” entry from a table specified by *table_name* . The order in which entries are returned is not guaranteed. Further, should an update occur in the table between client calls to *nis_next_entry()* there is no guarantee that an entry that is added or modified will be seen by the client. Should an entry be removed from the table that would have been the “next” entry returned, the error NIS_CHAINBROKEN is returned instead.

RETURN VALUES

These functions return a pointer to a structure of type *nis_result* :

```
struct nis_result {
    nis_error status;
    struct {
        uint_t objects_len;
        nis_object *objects_val;
    } objects;
    netobj cookie;
    uint32_t zticks;
    uint32_t dticks;
    uint32_t aticks;
    uint32_t cticks;
};
```

The *status* member contains the error status of the the operation. A text message that describes the error can be obtained by calling the function `nis_sperrno()` [see `nis_error(3N)`].

The *objects* structure contains two members. *objects_val* is an array of *nis_object* structures; *objects_len* is the number of cells in the array. These objects will be freed by a call to `nis_freeresult()` [see `nis_names(3N)`]. If you need to keep a copy of one or more objects, they can be copied with the function `nis_clone_object()` and freed with the function `nis_destroy_object()` [see `nis_server(3N)`].

The various ticks contain details of where the time (in microseconds) was taken during a request. They can be used to tune one's data organization for faster access and to compare different database implementations (see `nis_db(3N)`).

zticks The time spent in the NIS+ service itself, this count starts when the server receives the request and stops when it sends the reply.

dticks The time spent in the database backend, this time is measured from the time a database call starts, until a result is returned. If the request results in multiple calls to the database, this is the sum of all the time spent in those calls.

aticks The time spent in any "accelerators" or caches. This includes the time required to locate the server needed to resolve the request.

cticks The total time spent in the request, this clock starts when you enter the client library and stops when a result is returned. By subtracting the sum of the other ticks values from this value you can obtain the local overhead of generating a NIS+ request.

Subtracting the value in *dticks* from the value in *zticks* will yield the time spent in the service code itself. Subtracting the sum of the values in *zticks* and *aticks* from the value in *cticks* will yield the time spent in the client library itself. Note: all of the tick times are measured in microseconds.

ERRORS

The client library can return a variety of error returns and diagnostics. The more salient ones are documented below.

NIS_BADATTRIBUTE

The name of an attribute did not match up with a named column in the table, or the attribute did not have an associated value.

NIS_BADNAME

The name passed to the function is not a legal NIS+ name.

NIS_BADREQUEST

A problem was detected in the request structure passed to the client library.

NIS_CACHEEXPIRED

The entry returned came from an object cache that has *expired*. This means that the time to live value has gone to zero and the entry may have changed. If the flag NO_CACHE was passed to the lookup function then the lookup function will retry the operation to get an unexpired copy of the object.

NIS_CBERROR

An RPC error occurred on the server while it was calling back to the client. The transaction was aborted at that time and any unsent data was discarded.

NIS_CBRESULTS

Even though the request was successful, all of the entries have been sent to your callback function and are thus not included in this result.

NIS_FOREIGNNS

The name could not be completely resolved. When the name passed to the function would resolve in a namespace that is outside the NIS+ name tree, this error is returned with a NIS+ object of type DIRECTORY. The returned object contains the type of namespace and contact information for a server within that namespace.

NIS_INVALIDOBJ

The object pointed to by *object* is not a valid NIS+ entry object for the given table. This could occur if it had a mismatched number of columns, or a different data type (for example, binary or text) than the associated column in the table.

NIS_LINKNAMEERROR

The name passed resolved to a LINK type object and the contents of the object pointed to an invalid name.

NIS_MODFAIL

The attempted modification failed for some reason.

NIS_NAMEEXISTS

An attempt was made to add a name that already exists. To add the name, first remove the existing name and then add the new name or modify the existing named object.

NIS_NAMEUNREACHABLE

This soft error indicates that a server for the desired directory of the named table object could not be reached. This can occur when there is a network partition or the server has crashed. Attempting the operation again may succeed. See the HARD_LOOKUP flag.

NIS_NOCALLBACK

The server was unable to contact the callback service on your machine. This results in no data being returned.

NIS_NOMEMORY

Generally a fatal result. It means that the service ran out of heap space.

NIS_NOSUCHNAME

This hard error indicates that the named directory of the table object does not exist. This occurs when the server that should be the parent of the server that serves the table, does not know about the directory in which the table resides.

NIS_NOSUCHTABLE

The named table does not exist.

NIS_NOT_ME

A request was made to a server that does not serve the given name. Normally this will not occur, however if you are not using the built in location mechanism for servers, you may see this if your mechanism is broken.

NIS_NOTFOUND

No entries in the table matched the search criteria. If the search criteria was null (return all entries) then this result means that the table is empty and may safely be removed by calling the `nis_remove()`.

If the `FOLLOW_PATH` flag was set, this error indicates that none of the tables in the path contain entries that match the search criteria.

NIS_NOTMASTER

A change request was made to a server that serves the name, but it is not the master server. This can occur when a directory object changes and it specifies a new master server. Clients that have cached copies of the directory object in the `/var/nis/NIS_SHARED_DIRCACHE` file will need to have their cache managers restarted (use `nis_cachemgr -i`) to flush this cache.

NIS_NOTSAMEOBJ

An attempt to remove an object from the namespace was aborted because the object that would have been removed was not the same object that was passed in the request.

NIS_NOTSEARCHABLE

The table name resolved to a NIS+ object that was not searchable.

NIS_PARTIAL

This result is similar to `NIS_NOTFOUND` except that it means the request succeeded but resolved to zero entries. When this occurs, the server returns a copy of the table object instead of an entry so that the client may then process the path or implement some other local policy.

NIS_RPCERROR

This fatal error indicates the RPC subsystem failed in some way. Generally there will be a `syslog(3)` message indicating why the RPC request failed.

NIS_S_NOTFOUND

The named entry does not exist in the table, however not all tables in the path could be searched, so the entry may exist in one of those tables.

NIS_S_SUCCESS

Even though the request was successful, a table in the search path was not able to be searched, so the result may not be the same as the one you would have received if that table had been accessible.

NIS_SUCCESS

The request was successful.

NIS_SYSTEMERROR

Some form of generic system error occurred while attempting the request. Check the `syslog(3)` record for error messages from the server.

NIS_TOOMANYATTRS

The search criteria passed to the server had more attributes than the table had searchable columns.

NIS_TRYAGAIN

The server connected to was too busy to handle your request. `add_entry()`, `remove_entry()`, and `modify_entry()` return this error when the master server is currently updating its internal state. It can be returned to `nis_list()` when the function specifies a callback and the server does not have the resources to handle callbacks.

NIS_TYPEMISMATCH

An attempt was made to add or modify an entry in a table, and the entry passed was of a different type than the table.

ENVIRONMENT VARIABLES

NIS_PATH When set, this variable is the search path used by `nis_list()` if the flag `EXPAND_NAME` is set.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

WARNINGS

NOTES

To succeed, `nis_add_entry()`, `nis_remove_entry()`, and `nis_modify_entry()` must inherit the `PAF_TRUSTED_PATH` attribute.

`nis_cachemgr(1M)`, `nis_names(3N)`, `nis_server(3N)`,
`rpc_svc_calls(3N)`

`niscat(1)`, `niserror(1)`, `nismatch(1)`, `syslog(3)`,
`nis_clone_object(3N)`, `nis_db(3N)`, `nis_objects(3N)`,
`nis_destroy_object(3N)`, `nis_error(3N)`, `nis_getnames(3N)`,
`nis_local_names(3N)`, `nis_objects(3N)`, `attributes(5)`

Use the flag `HARD_LOOKUP` carefully since it can cause the application to block indefinitely during a network partition.

The path used when the flag `FOLLOW_PATH` is specified, is the one present in the *first* table searched. The path values in tables that are subsequently searched are ignored.

It is legal to call functions that would access the nameservice from within a list callback. However, calling a function that would itself use a callback, or calling `nis_list()` with a callback from within a list callback function is not currently supported.

There are currently no known methods for `nis_first_entry()` and `nis_next_entry()` to get their answers from only the master server.

The `nis_list()` function is not MT-Safe with callbacks. `nis_list()` callbacks are serialized. A call to `nis_list()` with a callback from within `nis_list()` will deadlock. `nis_list()` with a callback cannot be called from an rpc server. See `rpc_svc_calls(3N)`. Otherwise, this function is MT-Safe.

NAME	nis_groups, nis_ismember, nis_addmember, nis_removemember, nis_creategroup, nis_destroygroup, nis_verifygroup, nis_print_group_entry – NIS+ group manipulation functions
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpcsvc/nis.h> bool_t nis_ismember(nis_name <i>principal</i>, nis_name <i>group</i>); nis_error nis_addmember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_removemember(nis_name <i>member</i>, nis_name <i>group</i>); nis_error nis_creategroup(nis_name <i>group</i>, uint_t <i>flags</i>); nis_error nis_destroygroup(nis_name <i>group</i>); void nis_print_group_entry(nis_name <i>group</i>); nis_error nis_verifygroup(nis_name <i>group</i>);</pre>
DESCRIPTION	<p>These functions manipulate NIS+ groups. They are used by NIS+ clients and servers, and are the interfaces to the group authorization object.</p> <p>The names of NIS+ groups are syntactically similar to names of NIS+ objects but they occupy a separate namespace. A group named "a.b.c.d." is represented by a NIS+ group object named "a.groups_dir.b.c.d."; the functions described here all expect the name of the group, not the name of the corresponding group object.</p> <p>There are three types of group members:</p> <ul style="list-style-type: none"> ■ An <i>explicit</i> member is just a NIS+ principal-name, for example "wickedwitch.west.oz." ■ An <i>implicit</i> ("domain") member, written "*.west.oz.", means that all principals in the given domain belong to this member. No other forms of wildcarding are allowed: "wickedwitch.*.oz." is invalid, as is "wickedwitch.west.*.". Note that principals in subdomains of the given domain are <i>not</i> included. ■ A <i>recursive</i> ("group") member, written "@cowards.oz.", refers to another group; all principals that belong to that group are considered to belong here. <p>Any member may be made <i>negative</i> by prefixing it with a minus sign ('-'). A group may thus contain explicit, implicit, recursive, negative explicit, negative implicit, and negative recursive members.</p> <p>A principal is considered to belong to a group if it belongs to at least one non-negative group member of the group and belongs to no negative group members.</p> <p>The <code>nis_ismember()</code> function returns TRUE if it can establish that <i>principal</i> belongs to <i>group</i>; otherwise it returns FALSE.</p>

The `nis_addmember()` and `nis_removemember()` functions add or remove a member. They do not check whether the member is valid. The user must have read and modify rights for the group in question. To succeed, `nis_addmember()` and `nis_removemember()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_creategroup()` and `nis_destroygroup()` functions create and destroy group objects. The user must have create or destroy rights, respectively, for the `groups_dir` directory in the appropriate domain. The parameter *flags* to `nis_creategroup()` is currently unused and should be set to zero. To succeed, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

The `nis_print_group_entry()` function lists a group's members on the standard output.

The `nis_verifygroup()` function returns `NIS_SUCCESS` if the given group exists, otherwise it returns an error code.

EXAMPLES

EXAMPLE 1 Simple Memberships

Given a group `sadsouls.oz.` with members `tinman.oz.`, `lion.oz.`, and `scarecrow.oz.`, the function call

```
bool_var = nis_ismember("lion.oz.", "sadsouls.oz.");
```

will return 1 (TRUE) and the function call

```
bool_var = nis_ismember("toto.oz.", "sadsouls.oz.");
```

will return 0 (FALSE).

EXAMPLE 2 Implicit Memberships

Given a group `baddies.oz.` with members `wickedwitch.west.oz.` and `*.monkeys.west.oz.`, the function call

`bool_var = nis_ismember("hogan.monkeys.west.oz.", "baddies.oz.");` will return 1 (TRUE) because any principal from the `monkeys.west.oz.` domain belongs to the implicit group `*.monkeys.west.oz.`, but the function call

```
bool_var = nis_ismember("hogan.big.monkeys.west.oz.", "baddies.oz.");
```

will return 0 (FALSE).

EXAMPLE 3 Recursive Memberships

Given a group `goodandbad.oz.` with members `toto.kansas`, `@sadsouls.oz.`, and `@baddies.oz.`, and the groups `sadsouls.oz.` and `baddies.oz.` defined above, the function call

```
bool_var = nis_ismember("wickedwitch.west.oz.", "goodandbad.oz.");
```

will return 1 (TRUE), because `wickedwitch.west.oz.` is a member of the `baddies.oz.` group which is recursively included in the `goodandbad.oz.` group.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

To succeed, `nis_addmember()`, `nis_removemember()`, `nis_creategroup()` and `nis_destroygroup()` must inherit the `PAF_TRUSTED_PATH` attribute.

SEE ALSO

Trusted Solaris 7
Reference Manual

The `nis_groups` man page refers to Solaris man pages only.

SunOS 5.7 Reference
Manual

`nisgrpadm(1)`, `nis_objects(3N)`, `attributes(5)`

NOTES

These functions only accept fully-qualified NIS+ names.

A group is represented by a NIS+ object (see `nis_objects(3N)`) with a variant part that is defined in the `group_obj` structure. It contains the following fields:

```
uint_t gr_flags; /* Interpretation Flags
    (currently unused) */
struct {
    uint_t gr_members_len;
    nis_name *gr_members_val;
} gr_members; /* Array of members */
```

NIS+ servers and clients maintain a local cache of expanded groups to enhance their performance when checking for group membership. Should the membership of a group change, servers and clients with that group cached will not see the change until either the group cache has expired or it is explicitly flushed. A server's cache may be flushed programmatically by calling the `nis_servstate()` function with tag `TAG_GCACHE` and a value of 1.

There are currently no known methods for `nis_ismember()`, `nis_print_group_entry()`, and `nis_verifygroup()` to get their answers from only the master server.

NAME	plock – Lock or unlock into memory process, text, or data								
SYNOPSIS	<pre>#include <sys/lock.h> int plock(int op);</pre>								
DESCRIPTION	<p>The <code>plock()</code> function allows the calling process to lock or unlock into memory its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock). Locked segments are immune to all routine swapping. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege to succeed.</p> <p>The <code>plock()</code> function performs the function specified by <i>op</i>:</p> <table> <tr> <td><code>PROCLCK</code></td><td>Lock text and data segments into memory (process lock).</td></tr> <tr> <td><code>TXTLCK</code></td><td>Lock text segment into memory (text lock).</td></tr> <tr> <td><code>DATLOCK</code></td><td>Lock data segment into memory (data lock).</td></tr> <tr> <td><code>UNLOCK</code></td><td>Remove locks.</td></tr> </table>	<code>PROCLCK</code>	Lock text and data segments into memory (process lock).	<code>TXTLCK</code>	Lock text segment into memory (text lock).	<code>DATLOCK</code>	Lock data segment into memory (data lock).	<code>UNLOCK</code>	Remove locks.
<code>PROCLCK</code>	Lock text and data segments into memory (process lock).								
<code>TXTLCK</code>	Lock text segment into memory (text lock).								
<code>DATLOCK</code>	Lock data segment into memory (data lock).								
<code>UNLOCK</code>	Remove locks.								
RETURN VALUES	<p><code>plock()</code> returns:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and sets <code>errno</code> to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and sets <code>errno</code> to indicate the error.				
0	On success.								
-1	On failure, and sets <code>errno</code> to indicate the error.								
ERRORS	<p>The <code>plock()</code> function fails and does not perform the requested operation if:</p> <table> <tr> <td><code>EAGAIN</code></td><td>Not enough memory.</td></tr> <tr> <td><code>EINVAL</code></td><td>The <i>op</i> argument is equal to <code>PROCLCK</code> and a process lock, a text lock, or a data lock already exists on the calling process; the <i>op</i> argument is equal to <code>TXTLCK</code> and a text lock or a process lock already exists on the calling process; the <i>op</i> argument is equal to <code>DATLOCK</code> and a data lock or a process lock already exists on the calling process; or the <i>op</i> argument is equal to <code>UNLOCK</code> and no lock exists on the calling process.</td></tr> <tr> <td><code>EPERM</code></td><td>The process does not have sufficient privilege.</td></tr> </table>	<code>EAGAIN</code>	Not enough memory.	<code>EINVAL</code>	The <i>op</i> argument is equal to <code>PROCLCK</code> and a process lock, a text lock, or a data lock already exists on the calling process; the <i>op</i> argument is equal to <code>TXTLCK</code> and a text lock or a process lock already exists on the calling process; the <i>op</i> argument is equal to <code>DATLOCK</code> and a data lock or a process lock already exists on the calling process; or the <i>op</i> argument is equal to <code>UNLOCK</code> and no lock exists on the calling process.	<code>EPERM</code>	The process does not have sufficient privilege.		
<code>EAGAIN</code>	Not enough memory.								
<code>EINVAL</code>	The <i>op</i> argument is equal to <code>PROCLCK</code> and a process lock, a text lock, or a data lock already exists on the calling process; the <i>op</i> argument is equal to <code>TXTLCK</code> and a text lock or a process lock already exists on the calling process; the <i>op</i> argument is equal to <code>DATLOCK</code> and a data lock or a process lock already exists on the calling process; or the <i>op</i> argument is equal to <code>UNLOCK</code> and no lock exists on the calling process.								
<code>EPERM</code>	The process does not have sufficient privilege.								
SUMMARY OF TRUSTED SOLARIS CHANGES	To succeed, <code>plock()</code> must have <code>PRIV_SYS_CONFIG</code> in its set of effective privileges.								
USAGE	The <code>mlock(3C)</code> and <code>mlockall(3C)</code> functions are the preferred interfaces for process locking.								
SEE ALSO Trusted Solaris 7 Reference Manual	<code>exec(2)</code> , <code>fork(2)</code> , <code>mlock(3C)</code> , <code>mlockall(3C)</code>								

**SunOS 5.7 Reference
Manual**

exit(2), memcntl(2)

NAME priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number

SYNOPSIS cc [*flag...*] *file...* -ltsol [*library...*]

```
#include <tsol/priv.h>
priv_t str_to_priv(const char * priv_name);

char * priv_to_str(const priv_t priv_id);

char * str_to_priv_set(const char * priv_names, priv_set_t * priv_set, const char *
separators);

char * priv_set_to_str(priv_set_t * priv_set, char separator, char * buffer, int * buflen);

char * get_priv_text(const priv_t priv_id);
```

DESCRIPTION

priv_to_str() returns a pointer to the statically allocated, null-terminated privilege name specified by *priv_id*. If *priv_id* is an undefined privilege ID , the integer ordinal of *priv_id* is returned. If *priv_id* is greater than TSOL_MAX_PRIV , the maximum allowable privilege ID , a NULL is returned.

str_to_priv() returns the numeric privilege ID specified by the null-terminated privilege name *priv_name*. Privilege names can be specified in upper or lower case. An integer ordinal in the string is also acceptable.

priv_set_to_str() appends the name of each privilege in *priv_set* to a string to which the user-supplied *buffer* of length *buflen* points. Privilege names are separated by the *separator* character. Integer ordinals name the undefined privileges found in the privilege set. String *none* identifies an empty privilege set; and *all* , a full privilege set. Privilege names in the string are sorted in alphabetical order by localized sort.

Based on the token separators (*separators*), str_to_priv_set() breaks the *priv_names* string into tokens to be translated into a privilege set. Token *none* is translated to an empty privilege set; token *all* , to a full privilege set. The presence of token *none* overrides whatever precedes it. For example, the string *file_mac_read,file_mac_write,none,proc_nofloat* produces the same result as *proc_nofloat* alone. The constructed privilege set is stored in the *priv_set_t* buffer to which *priv_set* points.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

<code>get_priv_text()</code>	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
<code>priv_to_str()</code>	Returns a pointer to the translated privilege name string. The function returns <code>NULL</code> and sets <code>errno</code> on failure.
<code>str_to_priv()</code>	Returns the numeric privilege ID. The function returns <code>-1</code> and sets <code>errno</code> on failure.
<code>priv_set_to_str()</code>	Returns a pointer to the translated privilege names string. If the passed-in <i>buflen</i> is too small to hold the string, this routine stores the required buffer size into <i>buflen</i> and returns <code>NULL</code> . The function returns <code>NULL</code> and sets <code>errno</code> on failure. This function returns <code>-1</code> if the string cannot be translated or if an integer ordinal in the string is greater than <code>TSOL_MAX_PRIV</code> .
<code>str_to_priv_set()</code>	Returns <code>NULL</code> on success. If bad privilege names appear in the <i>priv_names</i> string, the function returns a pointer to the first privilege name that is not recognizable.

ERRORS

<code>priv_to_str()</code> may fail for this reason:	
<code>EINVAL</code>	The specified <i>priv_id</i> is greater than <code>TSOL_MAX_PRIV</code> .
<code>priv_set_to_str()</code> may fail for this reason:	
<code>EFAULT</code>	The specified <i>priv_set</i> is an invalid address.
<code>str_to_priv()</code> may fail for one of these reasons:	
<code>EINVAL</code>	The specified <i>priv_name</i> does not match any of the defined privilege names.
<code>EFAULT</code>	The specified <i>priv_name</i> is an invalid address.

NOTES

To use these routines, the program must be loaded with the Trusted Solaris library `libtsol` or `libtsol.so`.

SEE ALSO

Trusted Solaris 7
Reference Manual

`priv_desc(4)` `priv_name(4)`

NAME | priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number

SYNOPSIS | cc [flag...] file... -ltsol [library...]

```
#include <tsol/priv.h>
priv_t str_to_priv(const char * priv_name);

char * priv_to_str(const priv_t priv_id);

char * str_to_priv_set(const char * priv_names, priv_set_t * priv_set, const char *
separators);

char * priv_set_to_str(priv_set_t * priv_set, char separator, char * buffer, int * buflen);

char * get_priv_text(const priv_t priv_id);
```

DESCRIPTION

priv_to_str() returns a pointer to the statically allocated, null-terminated privilege name specified by *priv_id*. If *priv_id* is an undefined privilege ID , the integer ordinal of *priv_id* is returned. If *priv_id* is greater than TSOL_MAX_PRIV , the maximum allowable privilege ID , a NULL is returned.

str_to_priv() returns the numeric privilege ID specified by the null-terminated privilege name *priv_name*. Privilege names can be specified in upper or lower case. An integer ordinal in the string is also acceptable.

priv_set_to_str() appends the name of each privilege in *priv_set* to a string to which the user-supplied *buffer* of length *buflen* points. Privilege names are separated by the *separator* character. Integer ordinals name the undefined privileges found in the privilege set. String *none* identifies an empty privilege set; and *all* , a full privilege set. Privilege names in the string are sorted in alphabetical order by localized sort.

Based on the token separators (*separators*), str_to_priv_set() breaks the *priv_names* string into tokens to be translated into a privilege set. Token *none* is translated to an empty privilege set; token *all* , to a full privilege set. The presence of token *none* overrides whatever precedes it. For example, the string *file_mac_read,file_mac_write,none,proc_nofloat* produces the same result as *proc_nofloat* alone. The constructed privilege set is stored in the *priv_set_t* buffer to which *priv_set* points.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

<code>get_priv_text()</code>	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
<code>priv_to_str()</code>	Returns a pointer to the translated privilege name string. The function returns <code>NULL</code> and sets <code>errno</code> on failure.
<code>str_to_priv()</code>	Returns the numeric privilege ID. The function returns <code>-1</code> and sets <code>errno</code> on failure.
<code>priv_set_to_str()</code>	Returns a pointer to the translated privilege names string. If the passed-in <i>buflen</i> is too small to hold the string, this routine stores the required buffer size into <i>buflen</i> and returns <code>NULL</code> . The function returns <code>NULL</code> and sets <code>errno</code> on failure. This function returns <code>-1</code> if the string cannot be translated or if an integer ordinal in the string is greater than <code>TSOL_MAX_PRIV</code> .
<code>str_to_priv_set()</code>	Returns <code>NULL</code> on success. If bad privilege names appear in the <i>priv_names</i> string, the function returns a pointer to the first privilege name that is not recognizable.

ERRORS

<code>priv_to_str()</code> may fail for this reason:	
<code>EINVAL</code>	The specified <i>priv_id</i> is greater than <code>TSOL_MAX_PRIV</code> .
<code>priv_set_to_str()</code> may fail for this reason:	
<code>EFAULT</code>	The specified <i>priv_set</i> is an invalid address.
<code>str_to_priv()</code> may fail for one of these reasons:	
<code>EINVAL</code>	The specified <i>priv_name</i> does not match any of the defined privilege names.
<code>EFAULT</code>	The specified <i>priv_name</i> is an invalid address.

NOTES

To use these routines, the program must be loaded with the Trusted Solaris library `libtsol` or `libtsol.so`.

SEE ALSO

Trusted Solaris 7
Reference Manual

`priv_desc(4)` `priv_name(4)`

**SunOS 5.7 Reference
Manual**

attributes(5)

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profstr_t *getprofstrbyname(char *name, int src); profstr_t *getprofstr(int src); int putprofstr(profstr_t *res, int src); int setprofstr(int stayopen, int src); int endprofstr(int src); void free_profstr(profstr_t *profstr);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofstrbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofstr() , getprofstr() , and endprofstr() are used to enumerate profile entries from the database. setprofstr() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofstr() . A call to getprofstrbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofstr() .</p> <p>Successive calls to getprofstr() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofstr() may be called to indicate that the caller expects to do no further profile stry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile string retrieval functions after calling endprofstr() .</p> <p>The functions getprofstrbyname() and getprofstr() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profstr() should be used to free the pointers returned by either getprofstrbyname() or getprofstr() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p>

The parameter *src* may be set to any of `TSOL_SRC_FILES`, `TSOL_SRC_NISPLUS`, or `TSOL_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`. For most applications the *src* parameter should be set to `TSOL_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

The function `putprofstr()` replaces an existing profile entry or adds a new entry if the profile name does not already exist. Currently the *src* parameter is treated as `TSOL_SRC_NISPLUS`, forcing all information to be written to the NIS+ table. Use of files or of `nsswitch.conf`(4) may be supported in future releases.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofstr()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofstr()`, the threads enumerate disjoint subsets of the `tsolprof`(4) database.

User entries are represented by the struct `profstr_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profstr_t {
    char  name;      /* name of profile */
    char  desc;      /* description */
    char  auths;     /* comma separated list of authorization numbers */
    char  actions;   /* semicolon separated action descriptions */
    char  cmds;      /* semicolon separated command descriptions */
} profstr_t;
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The function `getprofstrbyname()` returns a pointer to a `profstr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofstr()` returns a pointer to a `profstr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The function `putprofstr()` returns 0 on success.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

Intro(3)

attributes(5)

Programs that use the interfaces described here cannot be linked statically since the implementation of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see *Intro(3)* , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file);</pre>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a <code>utmp</code> structure with the following members:</p> <pre>char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutent() The <code>getutent()</code> function reads in the next entry from a <code>utmp</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The <code>getutid()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The <code>getutline()</code> function searches forward from the current point in the <code>utmp</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>

pututline()	<p>The <code>pututline()</code> function writes the supplied <code>utmp</code> structure into the <code>utmp</code> file. It uses <code>getutid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututline()</code> will have searched for the proper entry using one of the these functions. If so, <code>pututline()</code> will not search. If <code>pututline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmp</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
setutent()	The <code>setutent()</code> function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.
endutent()	The <code>endutent()</code> function closes the currently open file.
utmpname()	The <code>utmpname()</code> function allows the user to change the name of the file examined, from <code>/var/adm/utmp</code> to any other file. It is most often expected that this other file will be <code>/var/adm/wtmp</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpname()</code> function does not open the file but closes the old file if it is currently open and saves the new file name.
RETURN VALUES	A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, <code>utmpname()</code> returns 0. Otherwise, it returns 1.
USAGE	<p>These functions use buffered standard I/O for input, but <code>pututline()</code> uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the <code>utmp</code> and <code>wtmp</code> files.</p> <p>Applications should not access the <code>utmp</code> or <code>utmpx</code> database files directly, but should use the functions described on the <code>getutxent(3C)</code> manual page to interact with these files. Using these extended API s will ensure that the <code>utmp</code> and <code>utmpx</code> databases are maintained consistently.</p>
FILES	<div> <div><code>/var/adm/utmp</code></div> <div>User access and accounting information (old format)</div> </div>

<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

`pututline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututline()` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` functions, if the user has just modified those contents and passed the pointer back to `pututline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmp *utmp, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfile, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

	information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)								
<code>getutmpx()</code>	The <code>getutmpx()</code> function copies the information stored in the members of the <code>utmp</code> structure to the corresponding members of the <code>utmpx</code> structure. (See <code>getutent(3C)</code> for <code>utmp</code> structure)								
<code>updwtmp()</code>	The <code>updwtmp()</code> function checks the existence of <code>wfile</code> and its parallel file, whose name is obtained by appending an “f3x” to <code>wfile</code> . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. <code>utmp</code> is written to <code>wfile</code> and the corresponding <code>utmpx</code> structure is written to the parallel file.								
<code>updwtmpx()</code>	The <code>updwtmpx()</code> function checks the existence of <code>wfilex</code> and its parallel file, whose name is obtained by truncating the final “f3x” from <code>wfilex</code> . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. <code>utmpx</code> is written to <code>wfilex</code> , and the corresponding <code>utmp</code> structure is written to the parallel file.								
RETURN VALUES	A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.								
USAGE	<p>These functions use buffered standard I/O for input, but <code>pututxline()</code> uses an unbuffered write to avoid race conditions between processes trying to modify the <code>utmpx</code> and <code>wtmpx</code> files.</p> <p>Applications should not access the <code>utmp</code> or <code>utmpx</code> database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the <code>utmp</code> and <code>utmpx</code> databases are maintained consistently.</p>								
FILES	<table> <tr> <td><code>/var/adm/utmp</code></td><td>User access and accounting information (old format)</td></tr> <tr> <td><code>/var/adm/utmpx</code></td><td>User access and accounting information (new format)</td></tr> <tr> <td><code>/var/adm/wtmp</code></td><td>History of user access and accounting information (old format)</td></tr> <tr> <td><code>/var/adm/wtmpx</code></td><td>History of user access and accounting information (new format)</td></tr> </table>	<code>/var/adm/utmp</code>	User access and accounting information (old format)	<code>/var/adm/utmpx</code>	User access and accounting information (new format)	<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)	<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)
<code>/var/adm/utmp</code>	User access and accounting information (old format)								
<code>/var/adm/utmpx</code>	User access and accounting information (new format)								
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)								
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)								
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	Unsafe								

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	randomword – Generate random pronounceable password						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library]</pre>						
DESCRIPTION	<pre>#include <tsol/tsol.h> int randomword(char *word, char *hyphenated_word, const unsigned short minlen, const unsigned short maxlen, const unsigned char *seed);</pre> <p>randomword() generates random pronounceable passwords using the FIPS 181 algorithm. Upon successful completion, <i>word</i> is replaced with a new password with a length between <i>minlen</i> and <i>maxlen</i> inclusive. <i>hyphenated_word</i> is a hyphenated version of <i>word</i> showing its pronunciation. If <i>seed</i> is non-NULL, it is a random number seed of eight significant characters. A good choice is the user's old password. Successive calls to randomword() by the same program should pass a null pointer for <i>seed</i> to produce new random passwords using the initial <i>seed</i>.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Unsafe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Unsafe						
RETURN VALUES	<p>randomword() returns:</p> <p>-1 If <i>minlen</i> > <i>maxlen</i> or <i>seed</i> has not been set.</p> <p>0 If <i>maxlen</i> is zero (0).</p> <p>>0 Length of the password <i>word</i> generated.</p>						
EXAMPLES	<p>EXAMPLE 1 Randomwork Example</p> <pre>char password[10]; char hyphen_password[20]; char seed[9]; int len; int i; printf("Please enter old password: "); fgets(seed, 9, stdin); len = randomword(password, hyphen_password, 6, 8, seed); printf("password %s is pronounced %s\n", password, hyphen_password); for (i = 1; i < 5; i++) { len = randomword(password, hyphen_password, 6, 8, (unsigned char *) 0); printf("password %s is pronounced %s\n", password, hyphen_password); }</pre>						
SEE ALSO							

Trusted Solaris 7 Reference Manual	Federal Information Processing Standards Publication 181, <i>Automated Password Generator</i> , 5 October 1993
SunOS 5.7 Reference Manual	attributes(5)

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

res_search

`res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

res_mkquery

The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <code><arpa/nameser.h></code>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

res_mkupdrec `res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *ttl*, and returns a pointer to a linked list *ns_updrec*. It returns `NULL` upon failure.

res_update `res_update()` takes a linked list of resource records *ns_updrec* as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

res_mkupdate `res_mkupdate()` creates update packets by running through the elements of the *ns_updrec* link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the *rdata* portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search` `res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery` The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <code><arpa/nameser.h></code>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

res_mkupdrec `res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *ttl*, and returns a pointer to a linked list `ns_updrec`. It returns `NULL` upon failure.

res_update `res_update()` takes a linked list of resource records `ns_updrec` as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

res_mkupdate `res_mkupdate()` creates update packets by running through the elements of the `ns_updrec` link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the `rdata` portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search` `res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery` The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <arpa/nameser.h>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

res_mkupdrec `res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *ttl*, and returns a pointer to a linked list *ns_updrec*. It returns `NULL` upon failure.

res_update `res_update()` takes a linked list of resource records *ns_updrec* as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

res_mkupdate `res_mkupdate()` creates update packets by running through the elements of the *ns_updrec* link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the *rdata* portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search` `res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery` The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <arpa/nameser.h>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

res_mkupdrec

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

res_update

`res_update()` takes a linked list of resource records `ns_updrec` as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

res_mkupdate

`res_mkupdate()` creates update packets by running through the elements of the `ns_updrec` link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the `rdata` portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES

`/etc/resolv.conf` Resolver configuration file

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search`

`res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery`

The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <code><arpa/nameser.h></code>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
<code>res_query</code>	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
<code>res_send</code>	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
<code>dn_expand</code>	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
<code>dn_comp</code>	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

`res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *ttl*, and returns a pointer to a linked list `ns_updrec`. It returns `NULL` upon failure.

`res_update()` takes a linked list of resource records `ns_updrec` as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

`res_mkupdate()` creates update packets by running through the elements of the `ns_updrec` link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the `rdata` portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search` `res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery` The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <arpa/nameser.h>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

res_mkupdrec

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

res_update

`res_update()` takes a linked list of resource records `ns_updrec` as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

res_mkupdate

`res_mkupdate()` creates update packets by running through the elements of the `ns_updrec` link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the `rdata` portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES

`/etc/resolv.conf` Resolver configuration file

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search` `res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery` The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <code><arpa/nameser.h></code>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

res_mkupdrec

res_update

res_mkupdate

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

`res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *ttl*, and returns a pointer to a linked list `ns_updrec`. It returns `NULL` upon failure.

`res_update()` takes a linked list of resource records `ns_updrec` as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

`res_mkupdate()` creates update packets by running through the elements of the `ns_updrec` link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the *rdata* portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES

`/etc/resolv.conf` Resolver configuration file

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

`res_search` `res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

`res_mkquery` The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <arpa/nameser.h>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

`res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *ttl*, and returns a pointer to a linked list `ns_updrec`. It returns `NULL` upon failure.

`res_update()` takes a linked list of resource records `ns_updrec` as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

`res_mkupdate()` creates update packets by running through the elements of the `ns_updrec` link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the `rdata` portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	resolver, res_init, res_mkquery, res_mkupdate, res_mkupdrec, res_query, res_search, res_send, res_update, dn_comp, dn_expand – Resolver routines
SYNOPSIS	<pre>cc [flags...] file ... -lresolv -lsocket -lnsl [library...] #include <sys/types.h> #include <netinet/in.h> #include <arpa/nameser.h> #include <resolv.h> int res_init(void); int res_mkquery(int op, const char * dname, int class, int type, const char * data, int datalen, struct rrec * newrr, uchar_t * buf, int buflen); int res_mkupdate(ns_updrec ** rrecp_in, uchar * buf, int length); ns_updrec * res_mkupdrec(int section, const char * dname, uint_t class, uint_t type, uint_t ttl); int res_query(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_search(const char * dname, int class, int type, uchar_t * answer, int anslen); int res_send(uchar_t * msg, int msglen, uchar_t * answer, int anslen); int res_update(ns_updrec * rrecp_in); int dn_comp(const char * exp_dn, uchar_t * comp_dn, int length, uchar_t ** dnptrs, uchar_t ** lastdnptr); int dn_expand(const uchar_t * msg, const uchar_t * eomorig, uchar_t * comp_dn, char exp_dn, int length);</pre>
DESCRIPTION	<p>These routines are used for making, sending, and interpreting query and reply messages passed to and from Internet domain name servers. The <code>res_update()</code> and <code>res_mkupdrec()</code> routines are used to dynamically update the name server with resource records.</p> <p>The global structure <code>_res</code> holds options and state information. Option values can be set to affect the collective behavior of groups of resolver library routines. However, most resolver library routines use reasonable defaults so that the explicit enabling of an option is rarely required.</p> <p>The library manual page entry for the resolver library includes public domain routines beyond those described here. See <code>libresolv(4)</code>. Those function names that are exported but are not explained here are lower-level routines called by these routines. Their direct use is discouraged. If you do make direct use of unsupported routines, you do so at considerable added risk and with no expectation of documentation or other support beyond that available publicly.</p>

Options for the resolver library are stored as a single bit mask containing the bitwise-OR sum of the options enabled. The options stored in `_res.options` are those defined in `<resolv.h>` and as follows. The field `_res.options` is a member of the `_res` structure.

<code>RES_INIT</code>	True if the initial name server address and default domain name are initialized, that is, <code>res_init()</code> has been called.
<code>RES_DEBUG</code>	Print debugging messages.
<code>RES_AAONLY</code>	Accept authoritative answers only. With this option, <code>res_send()</code> will continue until it finds an authoritative answer or finds an error. Currently this option is not implemented.
<code>RES_USEVC</code>	Use TCP connections for queries instead of UDP datagrams.
<code>RES_PRIMARY</code>	Query primary server only. This option is not implemented.
<code>RES_IGNTC</code>	Unused currently. Ignore truncation errors; that is, do not retry with TCP.
<code>RES_RECURSE</code>	Set the recursion-desired bit in queries. This is the default. <code>res_send()</code> does not do iterative queries and expects the name server to handle recursion.
<code>RES_DEFNAMES</code>	If set, <code>res_search()</code> appends the default domain name to single-component names (names that do not contain a dot). This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
<code>RES_DNSRCH</code>	Enables searching up through the current domain tree. If this option is set, <code>res_search()</code> searches for host names in the current domain and in parent domains. This is used by the standard host lookup routine <code>gethostbyname(3N)</code> . This option is enabled by default.
<code>RES_NOALIASES</code>	This option turns off the user level aliasing feature controlled by the <code>HOSTALIASES</code> environment variable. Network daemons should set this option.

`res_init`

If the system initialization file `resolv.conf` exists, `res_init()` reads it to get the default domain name, the search list, and the Internet address of the local name server or servers. See `resolv.conf(4)`. If no server is configured by the local `resolv.conf` file, `res_init()` tries to obtain name resolution services from the host on which it is running.

The `res_init()` function also sets the `RES_INIT` field of the `_res` global structure so that other service routines (`res_search()`) can determine for certain whether it needs to be called first before other processing begins.

In the absence of a `resolv.conf` configuration file, the current domain is either set to the value of the environmental variable `LOCALDOMAIN`, derived from the domain name or derived from the host name. See `domainname(1M)`. The current domain name as defined in the system initialization file `resolv.conf` can be overridden by the environment variable `LOCALDOMAIN`. This environment variable may contain several blank-separated tokens if you wish to override the *search list* on a per-process basis. This is similar to the search command in the configuration file. Another environment variable (`RES_OPTIONS`) can be set to override certain internal resolver options. Otherwise, these options are set by changing fields in the global `_res` structure or they are inherited from the configuration file's *options* command. The syntax of the `RES_OPTIONS` environment variable is explained in `resolv.conf(4)`.

The initializations performed by `res_init()` can be invoked explicitly with this function. However, they are normally performed automatically as a result of a call to one of the other resolver routines.

res_search

`res_search()` formulates and sends a normal query (`QUERY`) message, and stores the response in a buffer supplied by the caller.

The parameters *class* and *type* define the class and type of query. See `<arpa/nameser.h>`. The response is returned in the user-supplied buffer *answer*. `res_search()` returns the length of *answer* in *anslen*. Like the other resolver routines, `res_search()` calls `res_init()` if the `RES_INIT` flag is not enabled.

The `res_search()` function acts in a similar manner as `res_query()`, except that it can implement the default and search rules controlled by the `RES_DEFNAMES` and `RES_DNSRCH` options.

The parameter *dname* is the domain name. However, if *dname* consists of a single-component name and the `RES_DEFNAMES` flag is enabled (the default), *dname* is appended with the current domain name.

If the `RES_DNSRCH` flag is enabled, `res_search()` searches up the current domain tree until an answer has been retrieved or an unrecoverable error has been encountered. `res_search()` returns the first successful reply.

res_mkquery

The `res_mkquery()` function constructs a standard query message and places it in *buf*.

`res_mkquery()` returns the size of the query or `-1` if the query is larger than *buflen*. The *op* parameter is usually `QUERY`, but can be any of the query types defined in `<arpa/nameser.h>`.

The parameter *dname* is the domain name for the query.

	<p>The parameters <i>class</i> and <i>type</i> define the class and type of query (see <arpa/nameser.h>). The parameter <i>data</i> is the resource record; <i>datalen</i> is the length of the record.</p> <p><i>newrr</i> is unused and should be specified as a null pointer (0).</p>
res_query	<p>The <code>res_query()</code> function provides an interface to most of the server query mechanism. It constructs a query, sends it to the local server, awaits a response, and makes preliminary checks on the reply. The query requests information of the specified <i>type</i> and <i>class</i> for the specified fully-qualified domain name <i>dname</i>. The reply message is left in the <i>answer</i> buffer with length <i>anslen</i> supplied by the caller. The <code>res_mkquery()</code> and <code>res_send()</code> routines described elsewhere in this section are lower-level routines that <code>res_query()</code> calls.</p>
res_send	<p><code>res_send()</code> sends a pre-formatted query to name servers and returns an answer. It calls <code>res_init()</code> if <code>RES_INIT</code> is not set, send the query to the local name server, and handle timeouts and retries. <i>msg</i> is the query sent; <i>msglen</i> is its length. <i>answer</i> is the response returned. The length of the response is stored in <i>anslen</i>. <code>res_send()</code> returns the length of the response or -1 if there were errors.</p> <p>If the query is sent to a name server on a non-trusted host, <code>res_send()</code> and <code>res_search()</code> can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the <code>net_upgrade_sl</code>, <code>net_downgrade_sl</code>, and <code>net_mac_read</code> privileges, then <code>res_send()</code> and <code>res_search()</code> can communicate with the name server regardless of the sensitivity label of the non-trusted host where the name server resides.</p>
dn_expand	<p><code>dn_expand()</code> expands the compressed domain name given by the pointer <i>comp_dn</i> into a full domain name. Expanded names are converted to upper case. The compressed name is contained in a query or reply message; <i>msg</i> is a pointer to the beginning of that message. Expanded names are stored in the buffer referenced by the <i>exp_dn</i> buffer of size <i>length</i>, which should be large enough to hold the expanded result.</p> <p><code>dn_expand()</code> returns the size of the compressed name, or -1 if there was an error.</p>
dn_comp	<p><code>dn_comp()</code> compresses the domain name <i>exp_dn</i> and stores it in <i>comp_dn</i>. <code>dn_comp()</code> returns the size of the compressed name, or -1 if there were errors. <i>length</i> is the size of the array pointed to by <i>comp_dn</i>. <i>dnptrs</i> is a pointer to the head of the list of pointers to previously compressed names in the current message. The first pointer must point to the beginning of the message. The list ends with <code>NULL</code>. The limit to the array is specified by <i>lastdnptr</i>.</p> <p>A side effect of calling <code>dn_comp()</code> is to update the list of pointers for labels inserted into the message by <code>dn_comp()</code> as the name is compressed. However,</p>

if *lastdnptr* is `NULL`, `dn_comp()` does not update the list of labels. If *dnptrs* is `NULL`, names are not compressed.

`res_mkupdrec()` takes the elements of a resource record as its arguments, for instance, *section*, *domainname*, *class*, *type*, and *ttl*, and returns a pointer to a linked list `ns_updrec`. It returns `NULL` upon failure.

`res_update()` takes a linked list of resource records `ns_updrec` as its only argument and separates the records into groups such that all records in a group will belong to a single name server. It creates a dynamic update packet for each zone and sends it to the name server and awaits an answer. Upon success, `res_update()` returns the number of zones updated. It returns `<0` upon error.

`res_mkupdate()` creates update packets by running through the elements of the `ns_updrec` link list. It is called by `res_update()` to create packets for `res_send()` to send to the name server. `res_mkupdate()` returns the actual size of the packet, or

- 1 Error in reading a word or number in the `rdata` portion for update packets.
- 2 The length of the packet passed is insufficient.
- 3 The zone section is not the first section in the linked list, or the section order has a problem.
- 4 A number overflow.
- 5 Unknown operations or no records.

If an error occurs it could leave the zones being updated in an inconsistent state. For example, a record might be successfully added to the forward lookup zone but the corresponding PTR record might have failed to update in the reverse lookup tables.

FILES `/etc/resolv.conf` Resolver configuration file

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the query is sent to a name server on a non-trusted host, `res_send()` and `res_search()` can communicate with the name server if its host's default sensitivity label matches the sensitivity label of the process issuing the call. If the calling process is run with the `PRIV_NET_UPGRADE_SL`, `PRIV_NET_DOWNGRADE_SL`, and `PRIV_NET_MAC_READ` privileges, then `res_send()` and `res_search()` can communicate with the name server

regardless of the sensitivity label of the non-trusted host where the name server resides.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`in.named(1M)` , `nstest(1M)` , `resolv.conf(4)`

**SunOS 5.7 Reference
Manual**

`domainname(1M)` , `gethostbyname(3N)` , `libresolv(4)` , `attributes(5)`

Lottor, M., *Domain Administrators Operators Guide* , RFC 1033, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Concepts and Facilities* , RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification* , RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Partridge, Craig, *Mail Routing and the Domain System* , RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986. Stahl, M., *Domain Administrators Guide* , RFC 1032, SRI International, Menlo Park, Calif., November 1987.

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.

NAME	rpc – Library routines for remote procedure calls		
SYNOPSIS	<pre>cc [flags...] file ... -lnsl [library...] #include <rpc/rpc.h> #include <netconfig.h></pre>		
DESCRIPTION	<p>These routines allow C language programs to make procedure calls on other machines across a network. First, the client sends a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>All RPC routines require the header <code><rpc/rpc.h></code>. Routines that take a <code>netconfig</code> structure also require that <code><netconfig.h></code> be included. Applications using RPC and XDR routines should be linked with the <code>libnsl</code> library.</p>		
Multithread Considerations	<p>In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). Defining this flag enables a thread-specific version of <code>rpc_createerr</code>. See <code>rpc_clnt_create(3N)</code>.</p> <p>When used in multithreaded applications, client-side routines are MT-Safe. CLIENT handles can be shared between threads; however, in this implementation, requests by different threads are serialized (that is, the first request will receive its results before the second request is sent). See <code>rpc_clnt_create(3N)</code>.</p> <p>When used in multithreaded applications, server-side routines are usually Unsafe. In this implementation the service transport handle, <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. See <code>rpc_svc_create(3N)</code>. Therefore, this structure cannot be freely shared between threads that call functions that do this. Routines that are affected by this restriction are marked as unsafe for MT applications. See <code>rpc_svc_calls(3N)</code>.</p>		
Nettyp	<p>Some of the high-level RPC interface routines take a <i>nettype</i> string as one of the parameters (for example, <code>clnt_create()</code>, <code>svc_create()</code>, <code>rpc_reg()</code>, <code>rpc_call()</code>). This string defines a class of transports which can be used for a particular application.</p> <p><i>nettype</i> can be one of the following:</p> <table> <tr> <td><code>netpath</code></td><td>Choose from the transports which have been indicated by their token names in the <code>NETPATH</code> environment variable. If <code>NETPATH</code> is unset or NULL, it defaults to <code>visible</code>. <code>netpath</code> is the default <i>nettype</i>.</td></tr> </table>	<code>netpath</code>	Choose from the transports which have been indicated by their token names in the <code>NETPATH</code> environment variable. If <code>NETPATH</code> is unset or NULL, it defaults to <code>visible</code> . <code>netpath</code> is the default <i>nettype</i> .
<code>netpath</code>	Choose from the transports which have been indicated by their token names in the <code>NETPATH</code> environment variable. If <code>NETPATH</code> is unset or NULL, it defaults to <code>visible</code> . <code>netpath</code> is the default <i>nettype</i> .		

visible	Choose the transports which have the visible flag (v) set in the <code>/etc/netconfig</code> file.
circuit_v	This is same as <code>visible</code> except that it chooses only the connection oriented transports (semantics <code>tpi_cots</code> or <code>tpi_cots_ord</code>) from the entries in the <code>/etc/netconfig</code> file.
datagram_v	This is same as <code>visible</code> except that it chooses only the connectionless datagram transports (semantics <code>tpi_clts</code>) from the entries in the <code>/etc/netconfig</code> file.
circuit_n	This is same as <code>netpath</code> except that it chooses only the connection oriented datagram transports (semantics <code>tpi_cots</code> or <code>tpi_cots_ord</code>).
datagram_n	This is same as <code>netpath</code> except that it chooses only the connectionless datagram transports (semantics <code>tpi_clts</code>).
udp	This refers to Internet UDP.
tcp	This refers to Internet TCP.

If `nettype` is `NULL`, it defaults to `netpath`. The transports are tried in left to right order in the `NETPATH` variable or in top to down order in the `/etc/netconfig` file.

Derived Types

In a 64-bit environment, the derived types are defined as follows:

typedef	uint32_t	rpcprog_t;
typedef	uint32_t	rpcvers_t;
typedef	uint32_t	rpcproc_t;
typedef	uint32_t	rpcprot_t;
typedef	uint32_t	rpcport_t;
typedef	int32_t	rpc_inline_t;

In a 32-bit environment, the derived types are defined as follows:

typedef	unsigned long	rpcprog_t;
typedef	unsigned long	rpcvers_t;
typedef	unsigned long	rpcproc_t;
typedef	unsigned long	rpcprot_t;
typedef	unsigned long	rpcport_t;
typedef	long	rpc_inline_t;

Data Structures

Some of the data structures used by the RPC package are shown below.

The AUTH Structure

```

union des_block {
    struct {
        u_int32  high;
        u_int32  low;
    } key;
    char  c[8];
};
typedef union des_block des_block;
extern bool_t xdr_des_block();
/*
 * Authentication info. Opaque to client.
 */
struct opaque_auth {
    enum_t oa_flavor;          /* flavor of auth */
    caddr_t oa_base;           /* address of more auth stuff */
    uint_t oa_length;          /* not to exceed MAX_AUTH_BYTES */
};
/*
 * Auth handle, interface to client side authenticators.
 */
typedef struct {
    struct opaque_auth ah_cred;
    struct opaque_auth ah_verf;
    union des_block ah_key;
    struct auth_ops {
        void(*ah_nextverf)();
        int(*ah_marshall)();      /* nextverf & serialize */
        int(*ah_validate)();      /* validate verifier */
        int(*ah_refresh)();       /* refresh credentials */
        void(*ah_destroy)();      /* destroy this structure */
    } *ah_ops;
    caddr_t ah_private;
} AUTH;

```

The CLIENT Structure

```

/*
 * Client rpc handle.
 * Created by individual implementations.
 * Client is responsible for initializing auth.
 */
typedef struct {
    AUTH *cl_auth;              /* authenticator */
    struct clnt_ops {
        enum clnt_stat (*cl_call)();      /* call remote procedure */
        void (*cl_abort)();               /* abort a call */
        void (*cl_geterr)();              /* get specific error code */
        bool_t (*cl_freeres)();            /* frees results */
        void (*cl_destroy)();              /* destroy this structure */
        bool_t (*cl_control)();            /* the ioctl() of rpc */
        int (*cl_settimers)();             /* set rpc level timers */
    } *cl_ops;
}

```


**The SVCXPRT
Structure**

```

        caddr_t    cl_private;           /* private stuff */
        char        *cl_netid;           /* network identifier */
        char        *cl_tp;              /* device name */
    }    CLIENT;

enum xpirt_stat {
    XPRT_DIED,
    XPRT_MOREREQS,
    XPRT_IDLE
};
/*
 * Server side transport handle
 */
typedef struct {
    int      xp_fd;                      /* file descriptor for the
    ushort_t xp_port;                    /* obsolete */
    struct xp_ops {
        bool_t (*xp_rcv)(); /* receive incoming requests */
        enum xpirt_stat (*xp_stat)(); /* get transport status */
        bool_t (*xp_getargs)(); /* get arguments */
        bool_t (*xp_reply)(); /* send reply */
        bool_t (*xp_freeargs)(); /* free mem allocated
                                   for args */
        void (*xp_destroy)(); /* destroy this struct */
    } *xp_ops;
    int xp_addrlen;                      /* length of remote addr.
    Obsolete */
    char *xp_tp;                         /* transport provider device
    name */
    char *xp_netid;                      /* network identifier */
    struct netbuf xp_ltaddr;              /* local transport address */
    struct netbuf xp_rtaddr;              /* remote transport address */
    char xp_raddr[16];                   /* remote address. Obsolete */
    struct opaque_auth xp_verf;          /* raw response verifier */
    caddr_t xp_p1;                       /* private: for use
    by svc ops */
    caddr_t xp_p2;                       /* private: for use
    by svc ops */
    caddr_t xp_p3;                       /* private: for use
    by svc lib */
    int xp_type                          /* transport type */
}    SVCXPRT;

```

**The svc_req
Structure**

```

struct svc_req {
    rpcprog_t rq_prog;                  /* service program number */
    rpcvers_t rq_vers;                  /* service protocol version */
    rpcproc_t rq_proc;                  /* the desired procedure */
    struct opaque_auth rq_cred; /* raw creds from the wire */
    caddr_t rq_clntcred;                /* read only cooked cred */
    SVCXPRT *rq_xprt;                  /* associated transport */
}

```

The XDR Structure

```

};

/*
 * XDR operations.
 * XDR_ENCODE causes the type to be encoded into the stream.
 * XDR_DECODE causes the type to be extracted from the stream.
 * XDR_FREE can be used to release the space allocated by an XDR_DECODE
 * request.
 */
enum xdr_op {
    XDR_ENCODE=0,
    XDR_DECODE=1,
    XDR_FREE=2
};
/*
 * This is the number of bytes per unit of external data.
 */
#define BYTES_PER_XDR_UNIT (4)
#define RNDUP(x) (((x) + BYTES_PER_XDR_UNIT - 1) /
    BYTES_PER_XDR_UNIT) \ * BYTES_PER_XDR_UNIT)
/*
 * A xdrproc_t exists for each data type which is to be encoded or
 * decoded. The second argument to the xdrproc_t is a pointer to
 * an opaque pointer. The opaque pointer generally points to a
 * structure of the data type to be decoded. If this points to 0,
 * then the type routines should allocate dynamic storage of the
 * appropriate size and return it.
 * bool_t (*xdrproc_t)(XDR *, caddr_t *);
 */
typedef bool_t (*xdrproc_t)();
/*
 * The XDR handle.
 * Contains operation which is being applied to the stream,
 * an operations vector for the particular implementation
 */
typedef struct {
    enum xdr_op x_op; /* operation; fast additional param */
    struct xdr_ops {
        bool_t (*x_getlong)(); /* get long from underlying stream */
        bool_t (*x_putlong)(); /* put long to underlying stream */
        bool_t (*x_getbytes)(); /* get bytes from underlying stream */
        bool_t (*x_putbytes)(); /* put bytes to underlying stream */
        uint_t (*x_getpostn)(); /* returns bytes off from beginning */
        bool_t (*x_setpostn)(); /* reposition the stream */
        rpc_inline_t (*x_inline)(); /* buf quick ptr to buffered data */
        void (*x_destroy)(); /* free privates of this xdr_stream */
        bool_t (*x_control)(); /* changed/retrieve client object info*/
        bool_t (*x_getint32)(); /* get int from underlying stream */
        bool_t (*x_putint32)(); /* put int to underlying stream */
    } *x_ops;
};

```

```

caddr_t    x_public;          /* users' data */
caddr_t    x_priv             /* pointer to private data */
caddr_t    x_base;           /* private used for position info */
int         x_handy;          /* extra private word */
XDR;

```

Index to Routines

The following index lists RPC routines and the manual reference pages on which they are described:

RPC Routine	Manual Reference Page
auth_destroy()	rpc_clnt_auth(3N)
authdes_create()	rpc_soc(3N)
authdes_getucred()	secure_rpc(3N)
authdes_seccreate()	secure_rpc(3N)
authkerb_getucred()	kerberos_rpc(3N)
authkerb_seccreate()	kerberos_rpc(3N)
authnone_create()	rpc_clnt_auth(3N)
authsys_create()	rpc_clnt_auth(3N)
authsys_create_default()	rpc_clnt_auth(3N)
authunix_create()	rpc_soc(3N)
authunix_create_default()	rpc_soc(3N)
callrpc()	rpc_soc(3N)
clnt_broadcast()	rpc_soc(3N)
clnt_call()	rpc_clnt_calls(3N)
clnt_control()	rpc_clnt_create(3N)
clnt_create()	rpc_clnt_create(3N)
clnt_destroy()	rpc_clnt_create(3N)
clnt_dg_create()	rpc_clnt_create(3N)
clnt_freeres()	rpc_clnt_calls(3N)
clnt_geterr()	rpc_clnt_calls(3N)
clnt_pcreateerror()	rpc_clnt_create(3N)
clnt_perrno()	rpc_clnt_calls(3N)

clnt_perror()	rpc_clnt_calls(3N)
clnt_raw_create()	rpc_clnt_create(3N)
clnt_spcreateerror()	rpc_clnt_create(3N)
clnt_sperrno()	rpc_clnt_calls(3N)
clnt_sperror()	rpc_clnt_calls(3N)
clnt_tli_create()	rpc_clnt_create(3N)
clnt_tp_create()	rpc_clnt_create(3N)
clnt_udpcreate()	rpc_soc(3N)
clnt_vc_create()	rpc_clnt_create(3N)
clntraw_create()	rpc_soc(3N)
clnttcp_create()	rpc_soc(3N)
clntudp_bufcreate()	rpc_soc(3N)
get_myaddress()	rpc_soc(3N)
getnetname()	secure_rpc(3N)
host2netname()	secure_rpc(3N)
key_decryptsession()	secure_rpc(3N)
key_encryptsession()	secure_rpc(3N)
key_gendes()	secure_rpc(3N)
key_setsecret()	secure_rpc(3N)
netname2host()	
netname2user()	secure_rpc(3N)
pmap_getmaps()	rpc_soc(3N)
pmap_getport()	rpc_soc(3N)
pmap_rmtcall()	rpc_soc(3N)
pmap_set()	rpc_soc(3N)
pmap_unset()	rpc_soc(3N)
rac_drop()	rpc_rac(3N)
rac_poll()	rpc_rac(3N)
rac_recv()	rpc_rac(3N)

<code>rac_send()</code>	<code>rpc_rac(3N)</code>
<code>registerrpc()</code>	<code>rpc_soc(3N)</code>
<code>rpc_broadcast()</code>	<code>rpc_clnt_calls(3N)</code>
<code>rpc_broadcast_exp()</code>	<code>rpc_clnt_calls(3N)</code>
<code>rpc_call()</code>	<code>rpc_clnt_calls(3N)</code>
<code>rpc_reg()</code>	<code>rpc_svc_calls(3N)</code>
<code>svc_create()</code>	<code>rpc_svc_create(3N)</code>
<code>svc_destroy()</code>	<code>rpc_svc_create(3N)</code>
<code>svc_dg_create()</code>	<code>rpc_svc_create(3N)</code>
<code>svc_dg_enablecache()</code>	<code>rpc_svc_calls(3N)</code>
<code>svc_fd_create()</code>	<code>rpc_svc_create(3N)</code>
<code>svc_fds()</code>	<code>rpc_soc(3N)</code>
<code>svc_freeargs()</code>	<code>rpc_svc_reg(3N)</code>
<code>svc_getargs()</code>	<code>rpc_svc_reg(3N)</code>
<code>svc_getcaller()</code>	<code>rpc_soc(3N)</code>
<code>svc_getreq()</code>	<code>rpc_soc(3N)</code>
<code>svc_getreqset()</code>	<code>rpc_svc_calls(3N)</code>
<code>svc_getrpccaller()</code>	<code>rpc_svc_calls(3N)</code>
<code>svc_kerb_reg()</code>	<code>kerberos_rpc(3N)</code>
<code>svc_raw_create()</code>	<code>rpc_svc_create(3N)</code>
<code>svc_reg()</code>	<code>rpc_svc_calls(3N)</code>
<code>svc_register()</code>	<code>rpc_soc(3N)</code>
<code>svc_run()</code>	<code>rpc_svc_reg(3N)</code>
<code>svc_sendreply()</code>	<code>rpc_svc_reg(3N)</code>
<code>svc_tli_create()</code>	<code>rpc_svc_create(3N)</code>
<code>svc_tp_create()</code>	<code>rpc_svc_create(3N)</code>
<code>svc_unreg()</code>	<code>rpc_svc_calls(3N)</code>
<code>svc_unregister()</code>	<code>rpc_soc(3N)</code>
<code>svc_vc_create()</code>	<code>rpc_svc_create(3N)</code>

svcerr_auth()	rpc_svc_err(3N)
svcerr_decode()	rpc_svc_err(3N)
svcerr_noproc()	rpc_svc_err(3N)
svcerr_noprogram()	rpc_svc_err(3N)
svcerr_progvers()	rpc_svc_err(3N)
svcerr_systemerr()	rpc_svc_err(3N)
svcerr_weakauth()	rpc_svc_err(3N)
svcfld_create()	rpc_soc(3N)
svccraw_create()	rpc_soc(3N)
svctcp_create()	rpc_soc(3N)
svcudp_bufcreate()	rpc_soc(3N)
svcudp_create()	rpc_soc(3N)
user2netname()	secure_rpc(3N)
xdr_accepted_reply()	rpc_xdr(3N)
xdr_authsys_parms()	rpc_xdr(3N)
xdr_authunix_parms()	rpc_soc(3N)
xdr_callhdr()	rpc_xdr(3N)
xdr_callmsg()	rpc_xdr(3N)
xdr_opaque_auth()	rpc_xdr(3N)
xdr_rejected_reply()	rpc_xdr(3N)
xdr_replymsg()	rpc_xdr(3N)
xprt_register()	rpc_svc_calls(3N)
xprt_unregister()	rpc_svc_calls(3N)

FILES

/etc/netconfig Network configuration database

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

The `CLIENT` and `SVCXPRT` structures allow clients and servers to provide `t6attr_t` pointers to opaque structures for accessing security attributes on requests and replies. When a new `CLIENT` or `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client or server must use the `t6alloc_blk()` routine to allocate attribute control structures and set the `t6attr_t` pointers in the `CLIENT` or `SVCXPRT` structure. When `clnt_destroy()` or `svc_destroy()` is used to destroy a handle, the client or server should also use `t6free_blk()` to free any attribute control structures previously allocated for that handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`, `rpc_svc_create(3N)`,
`rpc_svc_reg(3N)`, `rpcbind(3N)`

**SunOS 5.7 Reference
Manual**

`getnetconfig(3N)`, `getnetpath(3N)`, `kerberos_rpc(3N)`,
`rpc_clnt_auth(3N)`, `rpc_svc_err(3N)`, `rpc_xdr(3N)`, `secure_rpc(3N)`,
`xdr(3N)`, `netconfig(4)`, `rpc(4)`, `attributes(5)`, `environ(5)`

NAME	<p>rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service</p>
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>ssvcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. <code>rpcbind</code> maintains a list of mappings between programs and their universal addresses. See <code>rpcbind(1M)</code>.</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps()</pre> <p>An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>This interface to the <code>rpcbind</code> service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the <code>PRIV_NET_MAC_READ</code> privilege, all mappings are returned regardless of their sensitivity labels. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <pre>rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple `[prognum , versnum , netconf \Rightarrow nc_netid]` and the address on the machine's `rpcbind` service. If `netconf` is `NULL`, `rpcb_unset()` destroys all mapping between the triple `[prognum , versnum , all-ports]` and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>ssvcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. <code>rpcbind</code> maintains a list of mappings between programs and their universal addresses. See <code>rpcbind(1M)</code>.</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps()</pre> <p>An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>This interface to the <code>rpcbind</code> service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the <code>PRIV_NET_MAC_READ</code> privilege, all mappings are returned regardless of their sensitivity labels. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <pre>rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple [*prognum* , *versnum* , *netconf* \Rightarrow *nc_netid*] and the address on the machine's `rpcbind` service. If *netconf* is `NULL`, `rpcb_unset()` destroys all mapping between the triple [*prognum* , *versnum* , *all-transport*] and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	<p>rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service</p>
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>ssvcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. <code>rpcbind</code> maintains a list of mappings between programs and their universal addresses. See <code>rpcbind(1M)</code>.</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps() An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine will return NULL if the remote <code>rpcbind</code> could not be contacted. This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings. rpcb_getallmaps() This interface to the <code>rpcbind</code> service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the <code>PRIV_NET_MAC_READ</code> privilege, all mappings are returned regardless of their sensitivity labels. This routine will return NULL if the remote <code>rpcbind</code> could not be contacted. rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple `[prognum , versnum , netconf \Rightarrow nc_netid]` and the address on the machine's `rpcbind` service. If `netconf` is `NULL`, `rpcb_unset()` destroys all mapping between the triple `[prognum , versnum , all-transport]` and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>ssvcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. <code>rpcbind</code> maintains a list of mappings between programs and their universal addresses. See <code>rpcbind(1M)</code>.</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps()</pre> <p>An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>This interface to the <code>rpcbind</code> service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the <code>PRIV_NET_MAC_READ</code> privilege, all mappings are returned regardless of their sensitivity labels. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <pre>rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and the address on the machine's `rpcbind` service. If *netconf* is `NULL`, `rpcb_unset()` destroys all mapping between the triple [*prognum*, *versnum*, *all-transport*] and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>ssvcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. rpcbind maintains a list of mappings between programs and their universal addresses. See rpcbind(1M) .</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps() An interface to the rpcbind service, which returns a list of the current RPC program-to-address mappings on <i>host</i> . It uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i> . This routine will return NULL if the remote rpcbind could not be contacted. This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings. rpcb_getallmaps() This interface to the rpcbind service returns a list of the current RPC program-to-address mappings on <i>host</i> . This interface uses the transport specified through <i>netconf</i> to contact the remote rpcbind service on <i>host</i> . This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the PRIV_NET_MAC_READ privilege, all mappings are returned regardless of their sensitivity labels. This routine will return NULL if the remote rpcbind could not be contacted. rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple `[prognum , versnum , netconf \Rightarrow nc_netid]` and the address on the machine's `rpcbind` service. If `netconf` is `NULL`, `rpcb_unset()` destroys all mapping between the triple `[prognum , versnum , all-transport]` and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>ssvcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. <code>rpcbind</code> maintains a list of mappings between programs and their universal addresses. See <code>rpcbind(1M)</code>.</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps()</pre> <p>An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>This interface to the <code>rpcbind</code> service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the <code>PRIV_NET_MAC_READ</code> privilege, all mappings are returned regardless of their sensitivity labels. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <pre>rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple [*prognum* , *versnum* , *netconf* \Rightarrow *nc_netid*] and the address on the machine's `rpcbind` service. If *netconf* is `NULL`, `rpcb_unset()` destroys all mapping between the triple [*prognum* , *versnum* , *all-transport*] and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

`void clnt_perrno(const enum clnt_stat stat);`

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

`void clnt_perror(const CLIENT *clnt, const char *s);`

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

`char *clnt_sperrno(const enum clnt_stat stat);`

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

`char *clnt_sperror(const CLIENT *clnt, const char *s);`

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

`enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char *nettype);`

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

`void clnt_perrno(const enum clnt_stat stat);`

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

`void clnt_perror(const CLIENT *clnt, const char *s);`

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

`char *clnt_sperrno(const enum clnt_stat stat);`

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

`char *clnt_sperror(const CLIENT *clnt, const char *s);`

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

`enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char *nettype);`

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the netconfig structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses AUTH_SYS credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the PRIV_NET_BROADCAST privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the PRIV_NET_BROADCAST privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>ssvcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. <code>rpcbind</code> maintains a list of mappings between programs and their universal addresses. See <code>rpcbind(1M)</code>.</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps()</pre> <p>An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>This interface to the <code>rpcbind</code> service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the <code>PRIV_NET_MAC_READ</code> privilege, all mappings are returned regardless of their sensitivity labels. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <pre>rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple [*prognum* , *versnum* , *netconf* \Rightarrow *nc_netid*] and the address on the machine's `rpcbind` service. If *netconf* is `NULL`, `rpcb_unset()` destroys all mapping between the triple [*prognum* , *versnum* , *all-transport*] and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	rpcbind, rpcb_getmaps, rpcb_getallmaps, rpcb_getaddr, rpcb_gettime, rpcb_rmtcall, rpcb_set, rpcb_unset – Library routines for RPC bind service
SYNOPSIS	<pre>#include <rpc/rpc.h> struct rpcblist * rpcb_getmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); struct tsol_rpcblist * rpcb_getallmaps(const struct netconfig * <i>netconf</i>, const char * <i>host</i>); bool_t rpcb_getaddr(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, struct netbuf * <i>svcaddr</i>, const char * <i>host</i>); bool_t rpcb_gettime(const char * <i>host</i>, time_t * <i>timep</i>); enum clnt_stat rpcb_rmtcall(const struct netconfig * <i>netconf</i>, const char * <i>host</i>, const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>, struct netbuf * <i>svcaddr</i>); bool_t rpcb_set(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>, const struct netbuf * <i>svcaddr</i>); bool_t rpcb_unset(const rpcprog_t <i>prognum</i>, const rpcvers_t <i>versnum</i>, const struct netconfig * <i>netconf</i>);</pre>
DESCRIPTION	<p>These routines allow client C programs to make procedure calls to the RPC binder service. <code>rpcbind</code> maintains a list of mappings between programs and their universal addresses. See <code>rpcbind(1M)</code>.</p>
Routines	<pre>#include <rpc/rpc.h> rpcb_getmaps()</pre> <p>An interface to the <code>rpcbind</code> service, which returns a list of the current RPC program-to-address mappings on <i>host</i>. It uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <p>This interface returns all the mappings at the client's sensitivity label, and all multilevel mappings.</p> <pre>rpcb_getallmaps()</pre> <p>This interface to the <code>rpcbind</code> service returns a list of the current RPC program-to-address mappings on <i>host</i>. This interface uses the transport specified through <i>netconf</i> to contact the remote <code>rpcbind</code> service on <i>host</i>. This routine returns all the mappings at sensitivity labels dominated by the client's sensitivity label and all multilevel mappings. If the client has the <code>PRIV_NET_MAC_READ</code> privilege, all mappings are returned regardless of their sensitivity labels. This routine will return <code>NULL</code> if the remote <code>rpcbind</code> could not be contacted.</p> <pre>rpcb_getaddr()</pre>

An interface to the `rpcbind` service, which finds the address of the service on *host* that is registered with program number *prognum*, version *versnum*, and speaks the transport protocol associated with *netconf*. The address found is returned in *svcaddr*. *svcaddr* should be preallocated. This routine returns `TRUE` if it succeeds. A return value of `FALSE` means that the mapping does not exist, that no mapping exists at the sensitivity label of the client and no multilevel mapping exists, or that the RPC system failed to contact the remote `rpcbind` service. In the last case, the global variable `rpc_createerr` contains the RPC status. See `rpc_clnt_create(3N)`.

If both a mapping at the sensitivity label of the client and a multilevel mapping exist, the mapping at the sensitivity label of the client is returned.

`rpcb_gettime()`

This routine returns the time on *host* in *timep*. If *host* is `NULL`, `rpcb_gettime()` returns the time on its own machine. This routine returns `TRUE` if it succeeds, `FALSE` if it fails. `rpcb_gettime()` can be used to synchronize the time between the client and the remote server. This routine is particularly useful for secure RPC.

`rpcb_rmtcall()`

An interface to the `rpcbind` service, which instructs `rpcbind` on *host* to make an RPC call on your behalf to a procedure on that host. The `netconfig` structure should correspond to a connectionless transport. The parameter **svcaddr* will be modified to the server's address if the procedure succeeds. See `rpc_call()` and `clnt_call()` in `rpc_clnt_calls(3N)` for the definitions of other parameters.

This procedure should normally be used for a "ping" and nothing else. This routine allows programs to do lookup and call, all in one step.

Note - Even if the server is not running `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Trusted Solaris Note: If there is no mapping at the sensitivity label of the client and no multilevel mapping, `rpcbind` does not return any error messages to the caller. In such a case, the caller times out.

Note: `rpcb_rmtcall()` is only available for connectionless transports.

`rpcb_set()`

An interface to the `rpcbind` service, which establishes a mapping between the triple [*prognum*, *versnum*, *netconf* \Rightarrow *nc_netid*] and *svcaddr* on the machine's `rpcbind` service. The value of *nc_netid* must correspond to a network identifier that is defined by the `netconfig` database.

If the client has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping is being established to a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. See also `svc_reg()` in `rpc_svc_calls(3N)`. If there already exists such an entry with `rpcbind`, `rpcb_set()` will fail.

`rpcb_unset()`

An interface to the `rpcbind` service, which destroys the mapping between the triple [*prognum* , *versnum* , *netconf* \Rightarrow *nc_netid*] and the address on the machine's `rpcbind` service. If *netconf* is `NULL`, `rpcb_unset()` destroys all mapping between the triple [*prognum* , *versnum* , *all-transport*] and the addresses on the machine's `rpcbind` service.

The `PRIV_NET_MAC_READ` privilege is required to delete a multilevel mapping. If the mapping being deleted is for a privileged port, the client must have the `PRIV_NET_PRIVADDR` privilege.

This routine returns `TRUE` if it succeeds, `FALSE` otherwise. Only the owner of the service or a process with the `PRIV_NET_SETID` privilege can destroy the mapping. See also `svc_unreg()` in `rpc_svc_calls(3N)`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the client or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is asserted when `rpcb_set()` is called, a multilevel mapping is created. To delete a multilevel mapping, `rpcb_unset()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpcb_set()` or `rpcb_unset()` calls that create or delete mappings for a privileged port.

The `PRIV_NET_SETID` privilege is required by `rpcb_unset()` for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`rpcbind(1M)`, `rpcinfo(1M)`, `rpc_clnt_calls(3N)`,
`rpc_clnt_create(3N)`, `rpc_svc_calls(3N)`

`attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

`void clnt_perrno(const enum clnt_stat stat);`

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

`void clnt_perror(const CLIENT *clnt, const char *s);`

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

`char *clnt_sperrno(const enum clnt_stat stat);`

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

`char *clnt_sperror(const CLIENT *clnt, const char *s);`

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

`enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char *nettype);`

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the netconfig structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses AUTH_SYS credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the PRIV_NET_BROADCAST privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the PRIV_NET_BROADCAST privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – Library routines for client side calls
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a request to the server. Upon receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends back a reply.</p> <p>The <code>clnt_call()</code>, <code>rpc_call()</code>, and <code>rpc_broadcast()</code> routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.</p> <p>Some of the routines take a <code>CLIENT</code> handle as one of the parameters. A <code>CLIENT</code> handle can be created by an RPC creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>).</p> <p>These routines are safe for use in multithreaded applications. <code>CLIENT</code> handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>enum clnt_stat clnt_call(CLIENT * <i>clnt</i>, const rpcproc_t <i>procnum</i>, const xdrproc_t <i>inproc</i>, const caddr_t <i>in</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>, const struct timeval <i>tout</i>);</p> <p>A function macro that calls the remote procedure <i>procnum</i> associated with the client handle, <i>clnt</i>, which is obtained with an RPC client creation routine such as <code>clnt_create()</code> (see <code>rpc_clnt_create(3N)</code>). The parameter <i>inproc</i> is the XDR function used to encode the procedure's parameters, and <i>outproc</i> is the XDR function used to decode the procedure's results; <i>in</i> is the address of the procedure's argument(s), and <i>out</i> is the address of where to place the result(s). <i>tout</i> is the time allowed for results to be returned, which is overridden by a time-out set explicitly through <code>clnt_control()</code>, see <code>rpc_clnt_create(3N)</code>.</p> <p>If the remote call succeeds, the status returned is <code>RPC_SUCCESS</code>, otherwise an appropriate status is returned.</p> <p>bool_t clnt_freeres(CLIENT * <i>clnt</i>, const xdrproc_t <i>outproc</i>, caddr_t <i>out</i>);</p> <p>A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter <i>out</i> is the address of the results, and <i>outproc</i> is the XDR routine describing the results. This routine returns 1 if the results were successfully freed, and 0 otherwise.</p> <p>void clnt_geterr(const CLIENT * <i>clnt</i>, struct rpc_err * <i>errp</i>);</p>

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

```
void clnt_perrno(const enum clnt_stat stat);
```

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance `rpc_call()`.

```
void clnt_perror(const CLIENT * clnt, const char * s);
```

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance `clnt_call()`.

```
char *clnt_sperrno(const enum clnt_stat stat);
```

Take the same arguments as `clnt_perrno()`, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

`clnt_sperrno()` is normally used instead of `clnt_perrno()` when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with `printf()` [see `printf(3S)`], or if a message format different than that supported by `clnt_perrno()` is to be used. Note: unlike `clnt_sperror()` and `clnt_spcreatererror()` [see `rpc_clnt_create(3N)`], `clnt_sperrno()` does not return pointer to static data so the result will not get overwritten on each call.

```
char *clnt_sperror(const CLIENT * clnt, const char * s);
```

Like `clnt_perror()`, except that (like `clnt_sperrno()`) it returns a string instead of printing to standard error. However, `clnt_sperror()` does not append a newline at the end of the message.

Warning: Returns pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
enum clnt_stat rpc_broadcast(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const caddr_t in, const xdrproc_t outproc, caddr_t out, const resultproc_t eachresult, const char * nettype);
```

Like `rpc_call()`, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is `NULL`, it defaults to "netpath". Each time it receives a response, this routine calls `eachresult()`, whose form is:

```
bool_t eachresult(caddr_t out, const struct netbuf *addr,
const struct netconfig *netconf);
```

where *out* is the same as *out* passed to `rpc_broadcast()`, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the `netconfig` structure of the transport on which the remote server responded. If `eachresult()` returns 0, `rpc_broadcast()` waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes.

`rpc_broadcast()` uses `AUTH_SYS` credentials by default [see `rpc_clnt_auth(3N)`].

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_broadcast_exp(const rpcprog_t prognum, const rpcvers_t
versnum, const rpcproc_t procnum, const xdrproc_t xargs, caddr_t argsp, const
xdrproc_t xresults, caddr_t resultsp, const resultproc_t eachresult, const int
inittime, const int waittime, const char * nettype);
```

Like `rpc_broadcast()`, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that `rpc_broadcast_exp()` waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

The process requires the `PRIV_NET_BROADCAST` privilege.

```
enum clnt_stat rpc_call(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const rpcproc_t procnum, const xdrproc_t inproc, const char *
in, const xdrproc_t outproc, char * out, const char * nettype);
```

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on `rpc(3N)`. This routine returns `RPC_SUCCESS` if it succeeds, or an appropriate status is returned. Use the `clnt_perrno()` routine to translate failure status into error messages.

Warning: `rpc_call()` uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel. `rpc_broadcast()` and `rpc_broadcast_exp()` require the `PRIV_NET_BROADCAST` privilege.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpc(3N)`, `rpc_clnt_create(3N)`, `libt6(3N)`, `t6alloc_blk(3N)`,
`t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`printf(3S)`, `rpc_clnt_auth(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a CLIENT handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the CLIENT data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code></p> <p>A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>


```

CLGET_VERS  rpcvers_t      get the RPC program's version
                    number associated with the
                    client handle
CLSET_VERS  rpcvers_t set the RPC program's version
                    number associated with the
                    client handle. This assumes
                    that the RPC server for this
                    new version is still listening
                    at the address of the previous
                    version.
CLGET_XID  uint32_t get the XID of the previous
                    remote procedure call
CLSET_XID  uint32_t set the XID of the next
                    remote procedure call
CLGET_PROG rpcprog_t get program number
CLSET_PROG rpcprog_t set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

`clnt_control()` returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host* , const rpcprog_t *prognum* , const rpcvers_t *versnum* , const char * *nettype*) ;

Generic client creation routine for program *prognum* and version *versnum* . *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

`clnt_create()` tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using `clnt_control()` . This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`).

CLIENT *clnt_create_timed(const char * *host* , const rpcprog_t *prognum* , const rpcvers_t *versnum* , const char * *nettype* , const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

`CLIENT *clnt_create_vers(const char * host, const rpcprog_t prognum, rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype);`
 Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns NULL. A default timeout is set and can be modified using `clnt_control()`. This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

`CLIENT *clnt_create_vers_timed(const char * host, const rpcprog_t prognum, rpcvers_t * vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char * nettype, const struct timeval * timeout);`

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

`void clnt_destroy(CLIENT * clnt);`

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt =>cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).

```
CLIENT *clnt_dg_create(const int fildes , const struct netbuf * svcaddr , const
rpcprog_t prognum , const rpcvers_t versnum , const uint_t sendsz , const uint_t
recvsz ) ;
```

This routine creates an RPC client for the remote program *prognum* and version *versnum* ; the client uses a connectionless transport. The remote program is located at address *svcaddr* . The parameter *fildes* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()` . The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum , const rpcvers_t versnum );
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum* . The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()` .

```
char *clnt_spcreateerror(const char * s);
```

Like `clnt_pcreateerror()` , except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildes , const struct netconfig * netconf , const
struct netbuf * svcaddr , const rpcprog_t prognum , const rpcvers_t versnum , const
uint_t sendsz , const uint_t recvsz );
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum* . The remote program is located at address *svcaddr* . If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL , RPC_UNKNOWNADDR error is set. *fildes* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host , const rpcprog_t prognum , const
rpcvers_t versnum , const struct netconfig * netconf ) ;
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host , const rpcprog_t prognum ,
const rpcvers_t versnum , const struct netconfig * netconf , const struct timeval
* timeout );
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev , const struct netbuf * svcaddr , const
rpcprog_t prognum , const rpcvers_t versnum , const uint_t sendsz , const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

```
struct rpc_createerr rpc_createerr;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_screateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – Library routines for dealing with creation and manipulation of CLIENT handles
DESCRIPTION	<p>RPC library routines allow C language programs to make procedure calls on other machines across the network. First a CLIENT handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.</p> <p>These routines are MT-Safe. In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag is defined, <code>rpc_createerr</code> becomes a macro which enables each thread to have its own <code>rpc_createerr</code> .</p> <p>Programs can retrieve network security attributes from incoming responses, and privileged programs can set the network security attributes on outgoing requests. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the CLIENT data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t clnt_control(CLIENT * <i>clnt</i> , const uint_t <i>req</i> , char * <i>info</i>) ;</code></p> <p>A function macro to change or retrieve various information about a client object. <i>req</i> indicates the type of operation, and <i>info</i> is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of <i>req</i> and their argument types and what they do are:</p> <pre>CLSET_TIMEOUT struct timeval * set total timeout CLGET_TIMEOUT struct timeval * get total timeout</pre> <p>If the timeout is set using <code>clnt_control()</code> , the timeout argument passed by <code>clnt_call()</code> is ignored in all subsequent calls. If the timeout value is set to 0 , <code>clnt_control()</code> immediately returns <code>RPC_TIMEDOUT</code> . Set the timeout parameter to 0 for batching calls.</p> <pre>CLGET_SERVER_ADDR struct netbuf * get server's address CLGET_SVC_ADDR struct netbuf * get server's address CLGET_FD int * get associated file descriptor CLSET_FD_CLOSE void close the file descriptor when destroying the client handle (see clnt_destroy()) CLSET_FD_NCLOSE void do not close the file descriptor when destroying the client handle</pre>

```

CLGET_VERS  rpcvers_t      get the RPC program's version
                    number associated with the
                    client handle
CLSET_VERS  rpcvers_t set the RPC program's version
                    number associated with the
                    client handle. This assumes
                    that the RPC server for this
                    new version is still listening
                    at the address of the previous
                    version.
CLGET_XID  uint32_t get the XID of the previous
                    remote procedure call
CLSET_XID  uint32_t set the XID of the next
                    remote procedure call
CLGET_PROG rpcprog_t get program number
CLSET_PROG rpcprog_t set program number

```

The following operations are valid for connectionless transports only:

```

CLSET_RETRY_TIMEOUT struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT struct timeval *    get the retry timeout

```

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

`clnt_control()` returns TRUE on success and FALSE on failure.

CLIENT *clnt_create(const char * *host* , const rpcprog_t *prognum* , const rpcvers_t *versnum* , const char * *nettype*) ;

Generic client creation routine for program *prognum* and version *versnum* . *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database.

`clnt_create()` tries all the transports of the *nettype* class available from the NETPATH environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using `clnt_control()` . This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`).

CLIENT *clnt_create_timed(const char * *host* , const rpcprog_t *prognum* , const rpcvers_t *versnum* , const char * *nettype* , const struct timeval * *timeout*);

Generic client creation routine which is similar to `clnt_create()` but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_timed()` call behaves exactly like the `clnt_create()` call.

`CLIENT *clnt_create_vers(const char *host, const rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype);`
 Generic client creation routine which is similar to `clnt_create()` but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* <= **vers_outp* <= *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns NULL. A default timeout is set and can be modified using `clnt_control()`. This routine returns NULL if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

Note: `clnt_create()` returns a valid client handle even if the particular version number supplied to `clnt_create()` is not registered with the `rpcbind` service. This mismatch will be discovered by a `clnt_call` later (see `rpc_clnt_calls(3N)`). However, `clnt_create_vers()` does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

`CLIENT *clnt_create_vers_timed(const char *host, const rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, char *nettype, const struct timeval *timeout);`

Generic client creation routine similar to `clnt_create_vers()` but with the additional parameter *timeout*, which specifies the maximum amount of time allowed for each transport class tried. In all other respects, the `clnt_create_vers_timed()` call behaves exactly like the `clnt_create_vers()` call.

`void clnt_destroy(CLIENT *clnt);`

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling `clnt_destroy()`. If the RPC library opened the associated file descriptor, or `CLSET_FD_CLOSE` was set using `clnt_control()`, the file descriptor will be closed.

The caller should call `auth_destroy(clnt ⇒ cl_auth)` (before calling `clnt_destroy()`) to destroy the associated AUTH structure (see `rpc_clnt_auth(3N)`).


```
CLIENT *clnt_dg_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz );
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by `clnt_call()` (see `clnt_call()` in `rpc_clnt_calls(3N)`). The retry time out and the total time out periods can be changed using `clnt_control()`. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns NULL if it fails.

```
void clnt_pcreateerror(const char * s);
```

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

```
CLIENT *clnt_raw_create(const rpcprog_t prognum, const rpcvers_t versnum);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see `svc_raw_create()` in `rpc_svc_create(3N)`). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns NULL if it fails. `clnt_raw_create()` should be called after `svc_raw_create()`.

```
char *clnt_spcreateerror(const char * s);
```

Like `clnt_pcreateerror()`, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

```
CLIENT *clnt_tli_create(const int fildev, const struct netconfig * netconf, const
struct netbuf * svcaddr, const rpcprog_t prognum, const rpcvers_t versnum, const
uint_t sendsz, const uint_t recvsz);
```

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is NULL and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is NULL, RPC_UNKNOWNADDR error is set. *fildev* is a file descriptor which may be

open, bound and connected. If it is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If *fildev* is `RPC_ANYFD` and *netconf* is `NULL`, a `RPC_UNKNOWNPROTO` error is set. If *fildev* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), `clnt_tli_create()` calls appropriate client creation routines. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure. The remote `rpcbind` service (see `rpcbind(1M)`) is not consulted for the address of the remote service.

```
CLIENT *clnt_tp_create(const char * host, const rpcprog_t prognum, const
rpcvers_t versnum, const struct netconfig * netconf);
```

Like `clnt_create()` except `clnt_tp_create()` tries only one transport specified through *netconf*.

`clnt_tp_create()` creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using `clnt_control()` calls. The remote `rpcbind` service on the host *host* is consulted for the address of the remote service. This routine returns `NULL` if it fails. The `clnt_pcreateerror()` routine can be used to print the reason for failure.

```
CLIENT *clnt_tp_create_timed(const char * host, const rpcprog_t prognum,
const rpcvers_t versnum, const struct netconfig * netconf, const struct timeval
* timeout);
```

Like `clnt_tp_create()` except `clnt_tp_create_timed()` has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the `clnt_tp_create_timed()` call behaves exactly like the `clnt_tp_create()` call.

```
CLIENT *clnt_vc_create(const int fildev, const struct netbuf * svcaddr, const
rpcprog_t prognum, const rpcvers_t versnum, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildev* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails.

The address *svcaddr* should not be `NULL` and should point to the actual address of the remote program. `clnt_vc_create()` does not consult the remote `rpcbind` service for this information.

```
struct rpc_createerr rpc_createerr;
```

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine `clnt_pcreateerror()` to print the reason for the failure.

In multithreaded applications, `rpc_createerr` becomes a macro which enables each thread to have its own `rpc_createerr`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `CLIENT` structure allows a client to provide `t6attr_t` pointers to opaque structures for accessing the security attributes of a reply or request. When a new `CLIENT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the client uses the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `CLIENT` structure. When `clnt_destroy()` is used to destroy a client handle, the client should also use `t6free_blk()` to free any attribute-control structures previously allocated for that client handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)`, `rpc(3N)`, `rpc_clnt_calls(3N)`, `rpc_svc_create(3N)`,
`libt6(3N)`, `t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpc_clnt_auth(3N)`, `svc_raw_create(3N)`, `attributes(5)`

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the `User` MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT *svc, const uint_t req, void *info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*

```
int svc_create(const void (* dispatch)(const struct svc_req *, const SVCXPRT *),
const rpcprog_t prognum, const rpcvers_t versnum, const char * nettype);
    svc_create() creates server handles for all the transports belonging
    to the class nettype.
```

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

```
void svc_destroy(SVCXPRT * xprt);
```

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

```
SVCXPRT *svc_dg_create(const int fildes, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildev* should be open and bound. The server is not registered with `rpcbind(1M)`.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildev*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns `NULL` if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see `clnt_raw_create()` in `rpc_clnt_create(3N)`). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns `NULL` if it fails, and an error message is logged.

Note: `svc_run()` should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildev*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildev* is the file descriptor on which the service is listening. If *fildev* is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildev* is bound to the address specified by *bindaddr*, otherwise *fildev* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

`svc_tp_create()` creates a server handle for the network specified by *netconf*, and registers itself with the `rpcbind` service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev , const uint_t sendsz , const uint_t rcvsvsz);`

This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N) , attributes(5)

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
DESCRIPTION	<p>These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code>], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (* procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char * nettype);</code></p> <p>Register program <i>prognum</i>, procedure <i>procname</i>, and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i>, version <i>versnum</i>, and procedure <i>procnum</i>, <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its static result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters while <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i>. See <code>rpc(3N)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p> <p><code>int svc_reg(const SVCXPRT * xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (* dispatch)(), const struct netconfig * netconf);</code></p> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i> \Rightarrow <i>nc_netid</i>] to <i>xpirt</i> \Rightarrow <i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p>

```
void svc_unreg(const rpcprog_t prognum , const rpcvers_t versnum );
```

Remove from the `rpcbind` service, all mappings of the triple [*prognum* , *versnum* , *all-transport*] to network address and all mappings within the RPC service package of the double [*prognum* , *versnum*] to dispatch routines.

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

The `PRIV_NET_SETID` privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

```
int svc_auth_reg(const int cred_flavor , const enum auth_stat (*handler)());
```

Registers the service authentication routine *handler* with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type *cred_flavor* . This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call `svc_auth_reg()` after registering the service and prior to calling `svc_run()` . When needed to process an RPC credential of type *cred_flavor* , the *handler* procedure will be called with two parameters (`struct svc_req * rqst` , `struct rpc_msg * msg`) and is expected to return a valid `enum auth_stat` value. There is no provision to change or delete an authentication handler once registered.

The `svc_auth_reg()` routine returns 0 if the registration is successful, 1 if *cred_flavor* already has an authentication handler registered for it, and -1 otherwise.

```
void xpirt_register(const SVCXPRT * xprt );
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3N)`). Service implementors usually do not need this routine.

```
void xpirt_unregister(const SVCXPRT * xprt );
```

Before an RPC service transport handle *xprt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable `svc_fdset` [see `rpc_svc_calls(3N)`]. Service implementors usually do not need this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is on when `rpc_reg()` or `rpc_svc()` is called, a multilevel mapping is created. To delete a multilevel mapping, `svc_unreg()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The `PRIV_NET_SETID` privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`inetd(1M)`, `rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`,
`rpc_svc_create(3N)`, `rpcbind(3N)`

`select(3C)`, `rpc_svc_err(3N)`, `attributes(5)`

NAME	sbartos, sbclartos, sbsartos, sbilartos, sbclearartos – Translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> char * sbclartos(const bclabel_t * <i>label</i>, const int <i>len</i>); char * sbsartos(const bslabel_t * <i>label</i>, const int <i>len</i>); char * sbclearartos(const bclear_t * <i>clearance</i>, const int <i>len</i>);</pre>
DESCRIPTION	<p>The calling process must have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges to perform label translation on labels that dominate the current process's sensitivity label.</p> <p>These functions translate binary labels into canonical strings that are clipped to the number of printable characters specified in <i>len</i>. Clipping is required if the number of characters of the translated string is greater than <i>len</i>. 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, and a clipped indicator, "–>", is prepended to information labels. However, information labels (<i>IL</i> s) are now obsolete. See NOTES below.</p> <p>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. In the case of a binary CMW label, the sensitivity label portion must also dominate the information label portion. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p><code>sbclartos()</code> translates a binary CMW label into a clipped string of the form:</p> <p><i>INFORMATION LABEL</i> [<i>SENSITIVITY LABEL</i>]</p> <p>using the long form of the words in the information and sensitivity labels, the long form of the classification name of the information label, and the short form of the classification name of the sensitivity label. The <i>INFORMATION LABEL</i> is clipped first until there is only one character of it remaining, then the <i>SENSITIVITY LABEL</i> is clipped. If <i>len</i> is less than the minimum number of characters, eight, the translation will fail. However, information labels (<i>IL</i> s) are now obsolete. See NOTES below.</p> <p><code>sbsartos()</code> 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 <i>len</i> is less than the minimum number of characters, three, the translation fails.</p> <p><code>sbclearartos()</code> translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is</p>

less than the minimum number of characters, three, the translation fails. The translation of a clearance may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file which may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

RETURN VALUES

These routines 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

sbcitos

Assume that a *CMW Label* is:

```
UNAVAILABLE LOWER DRAWER [UN TOP/MIDDLE/LOWER DRAWER]
```

when clipped to twelve characters it is:

```
->U [UN TO<-
```

sbsltos

Assume that a *Sensitivity Label* is:

```
UN TOP/MIDDLE/LOWER DRAWER
```

when clipped to ten characters it is:

```
UN TOP/M<-
```

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.

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.

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.
- Options related to information labels in the label_encodings(4) file can be ignored:

```
Markings Name= Marks;
Float Process Information Label;
```

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) ,
blmanifest(3) , blminmax(3) , blportion(3) , bltcolor(3) , bltos(3)
, bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) ,
labelvers(3) , stobl(3) , label_encodings(4)

Trusted Solaris Developer's Guide , *Trusted Solaris user's document set* , and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(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.

NAME	sbltos, sbcltos, sbsltos, sbiltos, sbcleartos – Translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> char * sbcltos(const bclabel_t * <i>label</i> , const int <i>len</i>); char * sbsltos(const bslabel_t * <i>label</i> , const int <i>len</i>); char * sbcleartos(const bclear_t * <i>clearance</i> , const int <i>len</i>);</pre>
DESCRIPTION	<p>The calling process must have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges to perform label translation on labels that dominate the current process's sensitivity label.</p> <p>These functions translate binary labels into canonical strings that are clipped to the number of printable characters specified in <i>len</i> . Clipping is required if the number of characters of the translated string is greater than <i>len</i> . 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, and a clipped indicator, “–>”, is prepended to information labels. However, information labels (<i>IL s</i>) are now obsolete. See NOTES below. 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. In the case of a binary CMW label, the sensitivity label portion must also dominate the information label portion. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p><code>sbcltos()</code> translates a binary CMW label into a clipped string of the form:</p> <pre>INFORMATION LABEL [SENSITIVITY LABEL]</pre> <p>using the long form of the words in the information and sensitivity labels, the long form of the classification name of the information label, and the short form of the classification name of the sensitivity label. The <i>INFORMATION LABEL</i> is clipped first until there is only one character of it remaining, then the <i>SENSITIVITY LABEL</i> is clipped. If <i>len</i> is less than the minimum number of characters, eight, the translation will fail. However, information labels (<i>IL s</i>) are now obsolete. See NOTES below.</p> <p><code>sbsltos()</code> 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 <i>len</i> is less than the minimum number of characters, three, the translation fails.</p> <p><code>sbcleartos()</code> translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is</p>

less than the minimum number of characters, three, the translation fails. The translation of a clearance may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file which may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

RETURN VALUES

These routines 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**sbcltos**

Assume that a *CMW Label* is:

```
UNAVAILABLE LOWER DRAWER [UN TOP/MIDDLE/LOWER DRAWER]
```

when clipped to twelve characters it is:

```
->U [UN TO<-
```

sbsltos

Assume that a *Sensitivity Label* is:

```
UN TOP/MIDDLE/LOWER DRAWER
```

when clipped to ten characters it is:

```
UN TOP/M<-
```

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.

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.

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.
- Options related to information labels in the label_encodings(4) file can be ignored:

```
Markings Name= Marks;
Float Process Information Label;
```

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) ,
blmanifest(3) , blminmax(3) , blportion(3) , bltcolor(3) , bltos(3)
, bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) ,
labelvers(3) , stobl(3) , label_encodings(4)

Trusted Solaris Developer's Guide , *Trusted Solaris user's document set* , and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(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.

NAME	sbiltos, sbcltos, sbsltos, sbiltos, sbcleartos – Translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> char * sbcltos(const bclabel_t * <i>label</i>, const int <i>len</i>); char * sbsltos(const bslabel_t * <i>label</i>, const int <i>len</i>); char * sbcleartos(const bclear_t * <i>clearance</i>, const int <i>len</i>);</pre>
DESCRIPTION	<p>The calling process must have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges to perform label translation on labels that dominate the current process's sensitivity label.</p> <p>These functions translate binary labels into canonical strings that are clipped to the number of printable characters specified in <i>len</i>. Clipping is required if the number of characters of the translated string is greater than <i>len</i>. 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, and a clipped indicator, "–>", is prepended to information labels. However, information labels (<i>IL</i> s) are now obsolete. See NOTES below.</p> <p>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. In the case of a binary CMW label, the sensitivity label portion must also dominate the information label portion. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p><code>sbcltos()</code> translates a binary CMW label into a clipped string of the form:</p> <p><i>INFORMATION LABEL</i> [<i>SENSITIVITY LABEL</i>]</p> <p>using the long form of the words in the information and sensitivity labels, the long form of the classification name of the information label, and the short form of the classification name of the sensitivity label. The <i>INFORMATION LABEL</i> is clipped first until there is only one character of it remaining, then the <i>SENSITIVITY LABEL</i> is clipped. If <i>len</i> is less than the minimum number of characters, eight, the translation will fail. However, information labels (<i>IL</i> s) are now obsolete. See NOTES below.</p> <p><code>sbsltos()</code> 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 <i>len</i> is less than the minimum number of characters, three, the translation fails.</p> <p><code>sbcleartos()</code> translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is</p>

less than the minimum number of characters, three, the translation fails. The translation of a clearance may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file which may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

RETURN VALUES

These routines 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

sbcltos

Assume that a *CMW Label* is:

```
UNAVAILABLE LOWER DRAWER [UN TOP/MIDDLE/LOWER DRAWER]
```

when clipped to twelve characters it is:

```
->U [UN TO<-
```

sbsltos

Assume that a *Sensitivity Label* is:

```
UN TOP/MIDDLE/LOWER DRAWER
```

when clipped to ten characters it is:

```
UN TOP/M<-
```

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.

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.

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.
- Options related to information labels in the label_encodings(4) file can be ignored:

```
Markings Name= Marks;
Float Process Information Label;
```

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) ,
blmanifest(3) , blminmax(3) , blportion(3) , bltcolor(3) , bltos(3)
, bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) ,
labelvers(3) , stobl(3) , label_encodings(4)

Trusted Solaris Developer's Guide , *Trusted Solaris user's document set* , and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(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.

NAME	sbltos, sbcltos, sbsltos, sbiltos, sbcleartos – Translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> char * sbcltos(const bclabel_t * <i>label</i> , const int <i>len</i>); char * sbsltos(const bslabel_t * <i>label</i> , const int <i>len</i>); char * sbcleartos(const bclear_t * <i>clearance</i> , const int <i>len</i>);</pre>
DESCRIPTION	<p>The calling process must have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges to perform label translation on labels that dominate the current process's sensitivity label.</p> <p>These functions translate binary labels into canonical strings that are clipped to the number of printable characters specified in <i>len</i> . Clipping is required if the number of characters of the translated string is greater than <i>len</i> . 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, and a clipped indicator, “–>”, is prepended to information labels. However, information labels (<i>IL s</i>) are now obsolete. See NOTES below. 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. In the case of a binary CMW label, the sensitivity label portion must also dominate the information label portion. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p><code>sbcltos()</code> translates a binary CMW label into a clipped string of the form:</p> <pre>INFORMATION LABEL [SENSITIVITY LABEL]</pre> <p>using the long form of the words in the information and sensitivity labels, the long form of the classification name of the information label, and the short form of the classification name of the sensitivity label. The <i>INFORMATION LABEL</i> is clipped first until there is only one character of it remaining, then the <i>SENSITIVITY LABEL</i> is clipped. If <i>len</i> is less than the minimum number of characters, eight, the translation will fail. However, information labels (<i>IL s</i>) are now obsolete. See NOTES below.</p> <p><code>sbsltos()</code> 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 <i>len</i> is less than the minimum number of characters, three, the translation fails.</p> <p><code>sbcleartos()</code> translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is</p>

less than the minimum number of characters, three, the translation fails. The translation of a clearance may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file which may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

RETURN VALUES

These routines 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 *CMW Label* is:

```
UNAVAILABLE LOWER DRAWER [UN TOP/MIDDLE/LOWER DRAWER]
```

when clipped to twelve characters it is:

```
-->U [UN TO<--
```

sbsltos

Assume that a *Sensitivity Label* is:

```
UN TOP/MIDDLE/LOWER DRAWER
```

when clipped to ten characters it is:

```
UN TOP/M<--
```

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.

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.

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.
- Options related to information labels in the label_encodings(4) file can be ignored:

```
Markings Name= Marks;
Float Process Information Label;
```

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) ,
blmanifest(3) , blminmax(3) , blportion(3) , bltcolor(3) , bltos(3)
, bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) ,
labelvers(3) , stobl(3) , label_encodings(4)

Trusted Solaris Developer's Guide , *Trusted Solaris user's document set* , and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(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.

NAME	sbltos, sbcltos, sbsltos, sbiltos, sbcleartos – Translate binary labels to canonical character-coded labels
SYNOPSIS	<pre>cc [flag...] file... -ltso1 [library...] #include <tsol/label.h> char * sbcltos(const bclabel_t * <i>label</i>, const int <i>len</i>); char * sbsltos(const bslabel_t * <i>label</i>, const int <i>len</i>); char * sbcleartos(const bclear_t * <i>clearance</i>, const int <i>len</i>);</pre>
DESCRIPTION	<p>The calling process must have <code>PRIV_SYS_TRANS_LABEL</code> in its set of effective privileges to perform label translation on labels that dominate the current process's sensitivity label.</p> <p>These functions translate binary labels into canonical strings that are clipped to the number of printable characters specified in <i>len</i>. Clipping is required if the number of characters of the translated string is greater than <i>len</i>. 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, and a clipped indicator, "–>", is prepended to information labels. However, information labels (<i>IL</i> s) are now obsolete. See NOTES below.</p> <p>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. In the case of a binary CMW label, the sensitivity label portion must also dominate the information label portion. A <i>len</i> of 0 (zero) returns the entire string with no clipping.</p> <p><code>sbcltos()</code> translates a binary CMW label into a clipped string of the form:</p> <p><i>INFORMATION LABEL</i> [<i>SENSITIVITY LABEL</i>]</p> <p>using the long form of the words in the information and sensitivity labels, the long form of the classification name of the information label, and the short form of the classification name of the sensitivity label. The <i>INFORMATION LABEL</i> is clipped first until there is only one character of it remaining, then the <i>SENSITIVITY LABEL</i> is clipped. If <i>len</i> is less than the minimum number of characters, eight, the translation will fail. However, information labels (<i>IL</i> s) are now obsolete. See NOTES below.</p> <p><code>sbsltos()</code> 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 <i>len</i> is less than the minimum number of characters, three, the translation fails.</p> <p><code>sbcleartos()</code> translates a binary clearance into a clipped string using the long form of the words and the short form of the classification name. If <i>len</i> is</p>

less than the minimum number of characters, three, the translation fails. The translation of a clearance may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file which may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	Unsafe

RETURN VALUES

These routines 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

sbsltos

Assume that a *CMW Label* is:

```
UNAVAILABLE LOWER DRAWER [UN TOP/MIDDLE/LOWER DRAWER]
```

when clipped to twelve characters it is:

```
->U [UN TO<-
```

sbsltos

Assume that a *Sensitivity Label* is:

```
UN TOP/MIDDLE/LOWER DRAWER
```

when clipped to ten characters it is:

```
UN TOP/M<-
```

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.

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.

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.
- Options related to information labels in the label_encodings(4) file can be ignored:

```
Markings Name= Marks;
Float Process Information Label;
```

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) ,
blmanifest(3) , blminmax(3) , blportion(3) , bltcolor(3) , bltos(3)
, bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) ,
labelvers(3) , stobl(3) , label_encodings(4)

Trusted Solaris Developer's Guide , *Trusted Solaris user's document set* , and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(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.

NAME	send, sendto, sendmsg – Send a message from a socket				
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <sys/socket.h> ssize_t send(int s, const void * msg, size_t len, int flags); ssize_t sendto(int s, const void * msg, size_t len, int flags, const struct sockaddr * to, socklen_t tolen); ssize_t sendmsg(int s, const struct msghdr * msg, int flags);</pre>				
DESCRIPTION	<p>send(), sendto(), and sendmsg() are used to transmit a message to another transport end-point. send() may be used only when the socket is in a <i>connected</i> state, while sendto() and sendmsg() may be used at any time. <i>s</i> is a socket created with socket(3N) .</p> <p>The address of the target is given by <i>to</i> with <i>tolen</i> specifying its size. The length of the message is given by <i>len</i> . If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.</p> <p>A return value of -1 indicates locally detected errors only. It does not implicitly mean the message was not delivered.</p> <p>If the socket does not have enough buffer space available to hold the message being sent, send() blocks, unless the socket has been placed in non-blocking I/O mode (see fcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:</p> <table> <tr> <td>MSG_OOB</td><td>Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.</td></tr> <tr> <td>MSG_DONTROUTE</td><td>The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.</td></tr> </table> <p>See recv(3N) , for a description of the msghdr structure.</p>	MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.
MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.				
RETURN VALUES	These calls return the number of bytes sent, or -1 if an error occurred.				
ERRORS	<p>The calls fail if:</p> <table> <tr> <td>EBADF</td><td><i>s</i> is an invalid file descriptor.</td></tr> </table>	EBADF	<i>s</i> is an invalid file descriptor.		
EBADF	<i>s</i> is an invalid file descriptor.				

EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL	<i>to</i> len is not the size of a valid address for the specified address family.
EMSGSIZE	The socket requires that message be sent atomically, and the message was too long.
ENOMEM	There was insufficient memory available to complete the operation.
ENOSR	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK	<i>s</i> is not a socket.
EWouldBlock	The socket is marked non-blocking and the requested operation would block.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the process calling these routines possesses the `PRIV_NET_REPLY_EQUAL` privilege, the packets the process sends will carry the same CMW label as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.

SEE ALSO

Trusted Solaris 7
Reference Manual

`fcntl(2)`, `write(2)`, `getsockopt(3N)`

SunOS 5.7 Reference
Manual

`poll(2)`, `select(3C)`, `recv(3N)`, `socket(3N)`, `attributes(5)`, `socket(5)`

NAME	send, sendto, sendmsg – Send a message from a socket				
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <sys/socket.h> ssize_t send(int s, const void * msg, size_t len, int flags); ssize_t sendto(int s, const void * msg, size_t len, int flags, const struct sockaddr * to, socklen_t tolen); ssize_t sendmsg(int s, const struct msghdr * msg, int flags);</pre>				
DESCRIPTION	<p>send(), sendto(), and sendmsg() are used to transmit a message to another transport end-point. send() may be used only when the socket is in a <i>connected</i> state, while sendto() and sendmsg() may be used at any time. <i>s</i> is a socket created with socket(3N) .</p> <p>The address of the target is given by <i>to</i> with <i>tolen</i> specifying its size. The length of the message is given by <i>len</i> . If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.</p> <p>A return value of -1 indicates locally detected errors only. It does not implicitly mean the message was not delivered.</p> <p>If the socket does not have enough buffer space available to hold the message being sent, send() blocks, unless the socket has been placed in non-blocking I/O mode (see fcntl(2)). The select(3C) or poll(2) call may be used to determine when it is possible to send more data.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:</p> <table> <tr> <td>MSG_OOB</td><td>Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.</td></tr> <tr> <td>MSG_DONTROUTE</td><td>The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.</td></tr> </table> <p>See recv(3N) , for a description of the msghdr structure.</p>	MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.
MSG_OOB	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.				
RETURN VALUES	These calls return the number of bytes sent, or -1 if an error occurred.				
ERRORS	<p>The calls fail if:</p> <table> <tr> <td>EBADF</td><td><i>s</i> is an invalid file descriptor.</td></tr> </table>	EBADF	<i>s</i> is an invalid file descriptor.		
EBADF	<i>s</i> is an invalid file descriptor.				

EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL	<i>to</i> len is not the size of a valid address for the specified address family.
EMSGSIZE	The socket requires that message be sent atomically, and the message was too long.
ENOMEM	There was insufficient memory available to complete the operation.
ENOSR	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK	<i>s</i> is not a socket.
EWouldBlock	The socket is marked non-blocking and the requested operation would block.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the process calling these routines possesses the `PRIV_NET_REPLY_EQUAL` privilege, the packets the process sends will carry the same CMW label as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.

SEE ALSO

Trusted Solaris 7
Reference Manual

`fcntl(2)`, `write(2)`, `getsockopt(3N)`

SunOS 5.7 Reference
Manual

`poll(2)`, `select(3C)`, `recv(3N)`, `socket(3N)`, `attributes(5)`, `socket(5)`

NAME	send, sendto, sendmsg – Send a message from a socket				
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <sys/socket.h> ssize_t send(int s, const void * msg, size_t len, int flags); ssize_t sendto(int s, const void * msg, size_t len, int flags, const struct sockaddr * to, socklen_t tolen); ssize_t sendmsg(int s, const struct msghdr * msg, int flags);</pre>				
DESCRIPTION	<p><code>send()</code>, <code>sendto()</code>, and <code>sendmsg()</code> are used to transmit a message to another transport end-point. <code>send()</code> may be used only when the socket is in a <i>connected</i> state, while <code>sendto()</code> and <code>sendmsg()</code> may be used at any time. <i>s</i> is a socket created with <code>socket(3N)</code>.</p> <p>The address of the target is given by <i>to</i> with <i>tolen</i> specifying its size. The length of the message is given by <i>len</i>. If the message is too long to pass atomically through the underlying protocol, then the error <code>EMSGSIZE</code> is returned, and the message is not transmitted.</p> <p>A return value of <code>-1</code> indicates locally detected errors only. It does not implicitly mean the message was not delivered.</p> <p>If the socket does not have enough buffer space available to hold the message being sent, <code>send()</code> blocks, unless the socket has been placed in non-blocking I/O mode (see <code>fcntl(2)</code>). The <code>select(3C)</code> or <code>poll(2)</code> call may be used to determine when it is possible to send more data.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of the following:</p> <table> <tr> <td><code>MSG_OOB</code></td><td>Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only <code>SOCK_STREAM</code> sockets created in the <code>AF_INET</code> address family support out-of-band data.</td></tr> <tr> <td><code>MSG_DONTROUTE</code></td><td>The <code>SO_DONTROUTE</code> option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.</td></tr> </table> <p>See <code>recv(3N)</code>, for a description of the <code>msghdr</code> structure.</p>	<code>MSG_OOB</code>	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only <code>SOCK_STREAM</code> sockets created in the <code>AF_INET</code> address family support out-of-band data.	<code>MSG_DONTROUTE</code>	The <code>SO_DONTROUTE</code> option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.
<code>MSG_OOB</code>	Send “out-of-band” data on sockets that support this notion. The underlying protocol must also support “out-of-band” data. Only <code>SOCK_STREAM</code> sockets created in the <code>AF_INET</code> address family support out-of-band data.				
<code>MSG_DONTROUTE</code>	The <code>SO_DONTROUTE</code> option is turned on for the duration of the operation. It is used only by diagnostic or routing programs.				
RETURN VALUES	These calls return the number of bytes sent, or <code>-1</code> if an error occurred.				
ERRORS	<p>The calls fail if:</p> <table> <tr> <td><code>EBADF</code></td><td><i>s</i> is an invalid file descriptor.</td></tr> </table>	<code>EBADF</code>	<i>s</i> is an invalid file descriptor.		
<code>EBADF</code>	<i>s</i> is an invalid file descriptor.				

EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL	<i>to</i> len is not the size of a valid address for the specified address family.
EMSGSIZE	The socket requires that message be sent atomically, and the message was too long.
ENOMEM	There was insufficient memory available to complete the operation.
ENOSR	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK	<i>s</i> is not a socket.
EWouldBlock	The socket is marked non-blocking and the requested operation would block.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the process calling these routines possesses the PRIV_NET_REPLY_EQUAL privilege, the packets the process sends will carry the same CMW label as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.

SEE ALSO

Trusted Solaris 7
Reference Manual

fcntl(2), write(2), getsockopt(3N)

SunOS 5.7 Reference
Manual

poll(2), select(3C), recv(3N), socket(3N), attributes(5), socket(5)

NAME	getacinfo, getacdir, getacflg, getacmin, getacna, setac, endac – Get audit control file information				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <bsm/libbsm.h> int getacdir(char * dir, int len); int getacmin(int * min_val); int getacflg(char * auditstring, int len); int getacna(char * auditstring, int len); void setac(void); void endac(void);</pre>				
DESCRIPTION	<p>When first called, <code>getacdir()</code> provides information about the first audit directory in the <code>audit_control</code> file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in <code>audit_control(4)</code>. The parameter <code>len</code> specifies the length of the buffer <code>dir</code>. On return, <code>dir</code> points to the directory entry.</p> <p><code>getacmin()</code> reads the minimum value from the <code>audit_control</code> file and returns the value in <code>min_val</code>. The minimum value specifies how full the file system to which the audit files are being written can get before the script <code>audit_warn(1M)</code> is invoked.</p> <p><code>getacflg()</code> reads the system audit value from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>.</p> <p><code>getacna()</code> reads the system audit value for non-attributable audit events from the <code>audit_control</code> file and returns the value in <code>auditstring</code>. The parameter <code>len</code> specifies the length of the buffer <code>auditstring</code>. Non-attributable events are events that cannot be attributed to an individual user. <code>inetd(1M)</code> and several other daemons record non-attributable events.</p> <p>Calling <code>setac</code> rewinds the <code>audit_control</code> file to allow repeated searches.</p> <p>Calling <code>endac</code> closes the <code>audit_control</code> file when processing is complete.</p>				
FILES	<table> <tr> <td><code>/etc/security/audit_control</code></td><td>Contains default parameters read by the audit daemon, <code>auditd(1M)</code>.</td></tr> </table>	<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .		
<code>/etc/security/audit_control</code>	Contains default parameters read by the audit daemon, <code>auditd(1M)</code> .				
RETURN VALUES	<p><code>getacdir()</code>, <code>getacflg()</code>, <code>getacna()</code> and <code>getacmin()</code> return:</p> <table> <tr> <td>0</td><td>on success.</td></tr> <tr> <td>-2</td><td>On failure and set <code>errno</code> to indicate the error.</td></tr> </table>	0	on success.	-2	On failure and set <code>errno</code> to indicate the error.
0	on success.				
-2	On failure and set <code>errno</code> to indicate the error.				

`getacmin()` and `getacflg()` return:

1 On EOF .

`getacdir()` returns:

-1 on EOF .

2 if the directory search had to start from the beginning because one of the other functions was called between calls to `getacdir()` .

These functions return:

-3 If the directory entry format in the `audit_control` file is incorrect.

`getacdir()` , `getacflg()` and `getacna()` return:

-3 If the input buffer is too short to accommodate the record.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe.

SUMMARY OF TRUSTED SOLARIS CHANGES

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_warn(1M)` , `inetd(1M)` , `audit_control(4)`

SunOS 5.7 Reference
Manual

`attributes(5)`

NAME	getauclassent, getauclassnam, setauclass, endauclass, getauclassnam_r, getauclassent_r – get audit_class entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_class_ent * getauclassnam(const char * name); struct au_class_ent * getauclassnam_r(au_class_ent_t * class_int, const char * name); struct au_class_ent * getauclassent(void); struct au_class_ent * getauclassent_r(au_class_ent_t * class_int); void setauclass(void); void endauclass(void);</pre>
DESCRIPTION	<p>getauclassent() and getauclassnam() each return an audit_class entry.</p> <p>getauclassnam() searches for an audit_class entry with a given class name <i>name</i> .</p> <p>getauclassent() enumerates audit_class entries: successive calls to getauclassent() will return either successive audit_class entries or NULL .</p> <p>setauclass() “rewinds” to the beginning of the enumeration of audit_class entries. Calls to getauclassnam() may leave the enumeration in an indeterminate state, so setauclass() should be called before the first getauclassent() .</p> <p>endauclass() may be called to indicate that audit_class processing is complete; the system may then close any open audit_class file, deallocate storage, and so forth.</p> <p>getauclassent_r() and getauclassnam_r() both return a pointer to an audit_class entry as do their similarly named counterparts. They each take an additional argument, a pointer to pre-allocated space for an au_class_ent_t , which is returned if the call is successful. To assure there is enough space for the information returned, the applications programmer should be sure to allocate AU_CLASS_NAME_MAX and AU_CLASS_DESC_MAX bytes for the ac_name and ac_desc elements of the au_class_ent_t data structure.</p> <p>The internal representation of an audit_user entry is an au_class_ent structure defined in <bsm/libbsm.h> with the following members:</p> <pre>char *ac_name; au_class_t ac_class; char *ac_desc;</pre>

RETURN VALUES

`getauclassnam()` and `getauclassnam_r()` return a pointer to a `struct au_class_ent` if they successfully locate the requested entry; otherwise they return `NULL`.

`getauclassent()` and `getauclassent_r()` return a pointer to a `struct au_class_ent` if they successfully enumerate an entry; otherwise they return `NULL`, indicating the end of the enumeration.

FILES

`/etc/security/audit_class` Maps audit class numbers to audit class names.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

All of the functions described in this man-page are MT-Safe except `getauclassent()` and `getauclassnam()`. The two functions, `getauclassent_r()` and `getauclassnam_r()` have the same functionality as the unsafe functions, but have a slightly different function call interface in order to make them MT-Safe.

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_class(4)`, `audit_event(4)`

SunOS 5.7 Reference
Manual

`attributes(5)`

NOTES

All information in the MT-unsafe versions are contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauevent, getauevnam, getauevnum, getauevnonam, setauevent, endauevent, getauevent_r, getauevnam_r, getauevnum_r – Get audit_event entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_event_ent *getauevent(void); struct au_event_ent *getauevnam(char * name); struct au_event_ent *getauevnum(au_event_t event_number); au_event_t *getauevnonam(char * event_name); void setauevent(void); void endauevent(void); struct au_event_ent *getauevent_r(au_event_ent_t * e); struct au_event_ent *getauevnam_r(au_event_ent_t * e, char * name); struct au_event_ent *getauevnum_r(au_event_ent_t * e, au_event_t event_number);</pre>
DESCRIPTION	<p>These interfaces document the programming interface for obtaining entries from the audit_event(4) file. getauevent(), getauevnam(), getauevnum(), getauevent_r(), getauevnam_r(), and getauevnum_r() each return a pointer to an audit_event structure.</p> <p>getauevent() and getauevent_r() enumerate audit_event entries; successive calls to these functions will return either successive audit_event entries or NULL.</p> <p>getauevnam() and getauevnam_r() search for an audit_event entry with a given event_name.</p> <p>getauevnum() and getauevnum_r() search for an audit_event entry with a given event_number.</p> <p>getauevnonam() searches for an audit_event entry with a given event_name and returns the corresponding event number.</p> <p>setauevent() “rewinds” to the beginning of the enumeration of audit_event entries. Calls to getauevnam(), getauevnum(), getauevnonum(), getauevnam_r(), or getauevnum_r() may leave the enumeration in an indeterminate state; setauevent() should be called before the first getauevent() or getauevent_r().</p> <p>endauevent() may be called to indicate that audit_event processing is complete; the system may then close any open audit_event file, deallocate storage, and so forth.</p>

The three functions `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` each take an argument `e` which is a pointer to an `au_event_ent_t`. This pointer is returned on a successful function call. To assure there is enough space for the information returned, the applications programmer should be sure to allocate `AU_EVENT_NAME_MAX` and `AU_EVENT_DESC_MAX` bytes for the `ae_name` and `ae_desc` elements of the `au_event_ent_t` data structure.

The internal representation of an `audit_event` entry is an `au_event_ent` structure defined in `<bsm/libbsm.h>` with the following members:

```
au_event_t    ae_number;char          *ae_name;char
*ae_desc;au_class_t    ae_class;
```

RETURN VALUES

`getauevent()`, `getauevnam()`, `getauevnum()`, `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` return a pointer to a `struct au_event_ent` if the requested entry is successfully located; otherwise it returns `NULL`.

`getauevnonam()` returns an event number of type `au_event_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating it could not find the requested event name.

FILES

<code>/etc/security/audit_event</code>	Maps audit event numbers to audit event names.
<code>/etc/passwd</code>	Stores user- ID to username mappings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
MT-Level	MT-Safe with exceptions.

The functions `getauevent()`, `getauevnam()`, and `getauevnum()` are not MT-Safe; however, there are equivalent functions: `getauevent_r()`, `getauevnam_r()`, and `getauevnum_r()` — all of which provide the same functionality and a MT-Safe function call interface.

SUMMARY OF TRUSTED SOLARIS CHANGES SEE ALSO

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

**Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

getauclassent(3), audit_class(4), audit_event(4)

getpwnam(3C), passwd(4), attributes(5)

All information for the functions `getauevent()`, `getauevnam()`, and `getauevnum()` is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

NAME	getauusernam, getauuserent, setauuser, endauuser – Get audit_user entry
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/libbsm.h> struct au_user_ent *getauusernam(const char * name); struct au_user_ent *getauuserent(void); void setauuser(void); void endauuser(void); struct au_user_ent *getauusernam_r(au_user_ent_t * u, const char * name); struct au_user_ent *getauuserent_r(au_user_ent_t * u);</pre>
DESCRIPTION	<p>The <code>getauuserent()</code>, <code>getauusernam()</code>, <code>getauuserent_r()</code>, and <code>getauusernam_r()</code> functions each return an <code>audit_user</code> entry.</p> <p>The <code>getauusernam()</code> and <code>getauusernam_r()</code> functions search for an <code>audit_user</code> entry with a given login name <i>name</i>.</p> <p>The <code>getauuserent()</code> and <code>getauuserent_r()</code> functions enumerate <code>audit_user</code> entries; successive calls to these functions will return either successive <code>audit_user</code> entries or <code>NULL</code>.</p> <p>The <code>setauuser()</code> function “rewinds” to the beginning of the enumeration of <code>audit_user</code> entries. Calls to <code>getauusernam()</code> and <code>getauusernam_r()</code> may leave the enumeration in an indeterminate state, so <code>setauuser()</code> should be called before the first call to <code>getauuserent()</code> or <code>getauuserent_r()</code>.</p> <p>The <code>endauuser()</code> function may be called to indicate that <code>audit_user</code> processing is complete; the system may then close any open <code>audit_user</code> file, deallocate storage, and so forth.</p> <p>The <code>getauuserent_r()</code> and <code>getauusernam_r()</code> functions both take an argument <i>u</i>, which is a pointer to an <code>au_user_ent</code>. This is the pointer that is returned on successful function calls.</p> <p>The internal representation of an <code>audit_user</code> entry is an <code>au_user_ent</code> structure defined in <code><bsm/libbsm.h></code> with the following members:</p> <pre>char *au_name;au_mask_t au_always;au_mask_t au_never;</pre>
RETURN VALUES	The <code>getauusernam()</code> function returns a pointer to an <code>au_user_ent</code> structure if it successfully locates the requested entry; otherwise it returns <code>NULL</code> .

SUMMARY OF TRUSTED SOLARIS CHANGES

ATTRIBUTES

The `getauuserent()` function returns a pointer to an `au_user_ent` structure if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The functionality described in this man page is available only if auditing has been enabled. By default, auditing is enabled in the Trusted Solaris environment.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe with exceptions.

FILES

`/etc/security/audit_user`

Stores per-user audit event mask.

`/etc/passwd`

Stores user-id to username mappings.

SEE ALSO

Trusted Solaris 7
Reference Manual

`audit_user(4)`

SunOS 5.7 Reference
Manual

`getpwnam(3C)`, `passwd(4)`, `attributes(5)`

NOTES

All information for the `getauuserent()` and `getauusernam()` functions is contained in a static area, which may be overwritten, so it must be copied if it is to be saved.

The `getauusernam()` and `getauuserent()` functions are not MT-safe. The `getauusernam_r()` and `getauuserent_r()` functions provide the same functionality with interfaces that are MT-Safe.

NAME	bltype, setbltype – Compare and set the type of binary label						
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>int bltype(const void * <i>label</i>, const unsigned char <i>type</i>);</pre> <pre>void setbltype(void * <i>label</i>, const unsigned char <i>type</i>);</pre>						
DESCRIPTION	<p>These functions compare and set the type of binary labels.</p> <p><code>bltype()</code> examines <i>label</i> to determine if it is of the specified type <i>type</i>.</p> <p><code>setbltype()</code> sets the type of <i>label</i> to the specified type <i>type</i>.</p> <p><i>type</i> may be one of:</p> <p><code>SUN_SL_ID</code> <i>label</i> is a defined binary sensitivity label.</p> <p><code>SUN_SL_UN</code> <i>label</i> is an undefined binary sensitivity label.</p> <p><code>SUN_CLR_ID</code> <i>label</i> is a defined binary clearance.</p> <p><code>SUN_CLR_UN</code> <i>label</i> is an undefined binary clearance.</p> <p><code>SUN_CMW_ID</code> <i>label</i> is a binary CMW label whose sensitivity label and information label portions may or may not be defined. (<code>bltype()</code> only.)</p>						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<code>bltype()</code> returns non-zero if <i>label</i> is of type <i>type</i> , otherwise zero is returned.						
SEE ALSO							
Trusted Solaris 7 Reference Manual	<p><code>bcltobanner(3)</code>, <code>bilconjoin(3)</code>, <code>blcompare(3)</code>, <code>blinset(3)</code>, <code>blmanifest(3)</code>, <code>blminmax(3)</code>, <code>blportion(3)</code>, <code>bltcolor(3)</code>, <code>bltos(3)</code>, <code>blvalid(3)</code>, <code>btohex(3)</code>, <code>hextob(3)</code>, <code>labelinfo(3)</code>, <code>labelvers(3)</code>, <code>sbltos(3)</code>, <code>stobl(3)</code></p> <p><i>Trusted Solaris Developer's Guide</i></p>						

SunOS 5.7 Reference
Manual**WARNINGS**

attributes(5)

- `bltype(&cmw_label, SUN_CMW_ID)` checks the existence of a binary CMW label structure and not the portions of the structure that contain defined labels.
- When attempting to determine the type of a label, rather than to verify that a specific label type is present, check `SUN_CMW_ID` first.
- `setbltype()` makes no checks on the structure it is setting or the type value.

NOTES

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as `ADMIN_LOW` .

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]` ; however, the IL component is fixed at `ADMIN_LOW` .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.

NAME	blportion, bcltosl, bcltoil, biltolev, getcsl, getcil, setcsl, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltsol [library...]</pre> <pre>#include <tsol/label.h> bslabel_t * bcltosl(bslabel_t * label); void getcsl(bslabel_t * destination_label, const bslabel_t * source_label); void setcsl(bslabel_t * destination_label, const bslabel_t * source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltosl() and bcltoil() provide a pointer to the sensitivity label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcsl() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcsl() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltosl() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blequal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), bltocolr(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	blportion, bcltosl, bcltoil, biltolev, getcs1, getcil, setcs1, setcil – Access binary label portions						
SYNOPSIS	<pre>cc [flag...] file ... -ltso1 [library...]</pre> <pre>#include <tso1/label.h> bslabel_t * bcltos1(bslabel_t * label); void getcs1(bslabel_t * destination_label, const bslabel_t * source_label); void setcs1(bslabel_t * destination_label, const bslabel_t * source_label);</pre>						
DESCRIPTION	<p>These functions provide pointers to, extract, and replace portions of binary labels.</p> <p>bcltos1() and bcltoil() provide a pointer to the sensitivity label and information label portion of the binary CMW label <i>label</i> respectively.</p> <p>getcs1() and getcil() copy the sensitivity label and information label portion of the binary CMW label <i>source_label</i> to the binary sensitivity label and binary information label <i>destination_label</i> respectively.</p> <p>setcs1() and setcil() replace the value of the sensitivity label and information label portion of the binary CMW label <i>destination_label</i> with the value of the binary sensitivity label and binary information label <i>source_label</i> respectively.</p>						
RETURN VALUES	bcltos1() and bcltoil() return a pointer to their respective label types.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
EXAMPLES	<p>CODE EXAMPLE 1 Comparing sensitivity labels</p> <p>The following example shows how to compare the sensitivity label portion of a binary CMW label with a file's binary sensitivity label.</p> <pre>blequal(bcltosl(&cmw_label), &file_sensitivity_label)</pre>						
SEE ALSO	<p>bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3), blmanifest(3), blminmax(3), bltocolo(3), bltos(3), bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3), labelvers(3), sbltos(3), stobl(3)</p> <p><i>Trusted Solaris Developer's Guide</i></p>						

**SunOS 5.7 Reference
Manual
NOTES****attributes(5)**

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	set_effective_priv, set_inheritable_priv, set_permitted_priv – Assign a privilege set for the current process						
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/priv.h> int set_effective_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]); int set_permitted_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]); int set_inheritable_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]);</pre>						
DESCRIPTION	<p>These routines, located in the Trusted Solaris library, assign the effective, inheritable, and permitted privilege sets, respectively, for the current process. These routines provide a user-friendly interface to the system call <code>setppriv(2)</code>. <i>op</i> is one of these operations:</p> <p>PRIV_ON Add the specified privilege ID s to the privilege set of the target process.</p> <p>PRIV_OFF Clear the specified privileges from the privilege set of the target process.</p> <p>PRIV_SET Add the specified privilege ID s to the privilege set of the target process and clear all other privileges.</p> <p><i>privno</i> indicates the count of privilege ID s that follow. The behavior of these routines is undefined if <i>privno</i> is less than zero. <i>priv_id</i> is a numerical privilege ID defined in <code><priv_names.h></code>.</p> <p>Note that if <i>op</i> is PRIV_SET and <i>privno</i> is 0 , the target privilege set is initialized to the empty set.</p>						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <p>0 On success.</p> <p>-1 On failure, and set <code>errno</code> to indicate the error.</p>						
ERRORS	<p>EINVAL The specified privilege is invalid.</p>						

EPERM	A specified privilege is not permitted in the asserted set of privileges.
-------	---

SEE ALSO**Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual**

setppriv(2)

attributes(5)

NAME	set_effective_priv, set_inheritable_priv, set_permitted_priv – Assign a privilege set for the current process						
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/priv.h> int set_effective_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]); int set_permitted_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]); int set_inheritable_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]);</pre>						
DESCRIPTION	<p>These routines, located in the Trusted Solaris library, assign the effective, inheritable, and permitted privilege sets, respectively, for the current process. These routines provide a user-friendly interface to the system call <code>setppriv(2)</code>. <i>op</i> is one of these operations:</p> <p>PRIV_ON Add the specified privilege ID s to the privilege set of the target process.</p> <p>PRIV_OFF Clear the specified privileges from the privilege set of the target process.</p> <p>PRIV_SET Add the specified privilege ID s to the privilege set of the target process and clear all other privileges.</p> <p><i>privno</i> indicates the count of privilege ID s that follow. The behavior of these routines is undefined if <i>privno</i> is less than zero. <i>priv_id</i> is a numerical privilege ID defined in <code><priv_names.h></code>.</p> <p>Note that if <i>op</i> is PRIV_SET and <i>privno</i> is 0 , the target privilege set is initialized to the empty set.</p>						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <p>0 On success.</p> <p>-1 On failure, and set <code>errno</code> to indicate the error.</p>						
ERRORS	<p>EINVAL The specified privilege is invalid.</p>						

E <code>PERM</code>	A specified privilege is not permitted in the asserted set of privileges.
---------------------	---

SEE ALSO**Trusted Solaris 7
Reference Manual**

setppriv(2)

**SunOS 5.7 Reference
Manual**

attributes(5)

NAME	set_effective_priv, set_inheritable_priv, set_permitted_priv – Assign a privilege set for the current process						
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/priv.h> int set_effective_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]); int set_permitted_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]); int set_inheritable_priv(priv_op_t op, int privno, [, priv_t priv_id, ...]);</pre>						
DESCRIPTION	<p>These routines, located in the Trusted Solaris library, assign the effective, inheritable, and permitted privilege sets, respectively, for the current process. These routines provide a user-friendly interface to the system call <code>setppriv(2)</code>. <i>op</i> is one of these operations:</p> <p>PRIV_ON Add the specified privilege ID s to the privilege set of the target process.</p> <p>PRIV_OFF Clear the specified privileges from the privilege set of the target process.</p> <p>PRIV_SET Add the specified privilege ID s to the privilege set of the target process and clear all other privileges.</p> <p><i>privno</i> indicates the count of privilege ID s that follow. The behavior of these routines is undefined if <i>privno</i> is less than zero. <i>priv_id</i> is a numerical privilege ID defined in <code><priv_names.h></code>.</p> <p>Note that if <i>op</i> is PRIV_SET and <i>privno</i> is 0 , the target privilege set is initialized to the empty set.</p>						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>These routines return:</p> <p>0 On success.</p> <p>–1 On failure, and set <code>errno</code> to indicate the error.</p>						
ERRORS	<p>EINVAL The specified privilege is invalid.</p>						

EPERM	A specified privilege is not permitted in the asserted set of privileges.
-------	---

SEE ALSO**Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual**

setppriv(2)

attributes(5)

NAME	getprofent, setprofent, endprofent, getprofentbyname, free_profent – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profent_t *getprofentbyname(char * name, int src); profent_t *getprofent(int src); void setprofent(int stayopen , int src); void endprofent(int src); void free_profent(profent_t * profent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofentbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofent() , getprofent() , and endprofent() are used to enumerate profile entries from the database. setprofent() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofent() . A call to getprofentbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofent() .</p> <p>Successive calls to getprofent() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofent() may be called to indicate that the caller expects to do no further profile entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile entry retrieval functions after calling endprofent() .</p> <p>The functions getprofentbyname() and getprofent() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profent() should be used to free the pointers returned by either getprofentbyname() or getprofent() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p> <p>The parameter <i>src</i> may be set to any of TSOL_DB_SRC_FILES , TSOL_DB_SRC_NISPLUS , or TSOL_DB_SRC_SWITCH , which are defined</p>

in `<tsol/tsol.h>` . For most applications the `src` parameter should be set to `TSOL_DB_SRC_SWITCH` , indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_DB_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_DB_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofent()` , the threads will enumerate disjoint subsets of the `tsolprof` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the `struct profent_t` structure defined in `<tsol/prof.h>` :

```
typedef struct profent_t {
    char *name;      /* name of profile */
    char *desc;      /* description */
    char *auths;     /* comma separated list of authorization numbers */
    profact_t *actions; /* linked list of actions */
    profcmd_t *cmds;  /* linked list of commands */
} profent_t;
```

The function `getprofentbyname()` returns a pointer to a `struct profent_t` if it successfully locates the requested entry; otherwise it returns `NULL` .

The function `getprofent()` returns a pointer to a `struct profent_t` if it successfully enumerates an entry; otherwise it returns `NULL` , indicating the end of the enumeration.

ERRORS

The functions `getprofentbyname()` and `getprofent()` return `NULL` on a failure.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)` ,
Notes On Multithread Applications , for information about the use of the
`_REENTRANT` flag.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

`Intro(3)`

`attributes(5)`

NAME	getprofstr, putprofstr, setprofstr, endprofstr, getprofstrbyname, free_profstr – Get Trusted Solaris user profile description
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/prof.h> profstr_t *getprofstrbyname(char * name , int src); profstr_t *getprofstr(int src); int putprofstr(profstr_t * res , int src); int setprofstr(int stayopen , int src); int endprofstr(int src); void free_profstr(profstr_t * profstr);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user profiles from the tsolprof NIS+ database or the /etc/security/tsol/tsoluser file.</p> <p>getprofstrbyname() searches for information for a profile with the specified profile name given by the parameter <i>name</i> .</p> <p>The functions setprofstr() , getprofstr() , and endprofstr() are used to enumerate profile entries from the database. setprofstr() sets (or resets) the enumeration to the beginning of the set of Trusted Solaris profile entries. This function should be called before the first call to getprofstr() . A call to getprofstrbyname() leaves the enumeration position in an indeterminate state. If the <i>stayopen</i> flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to endprofstr() .</p> <p>Successive calls to getprofstr() return either successive entries or return NULL , indicating the end of the enumeration.</p> <p>endprofstr() may be called to indicate that the caller expects to do no further profile stry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more profile string retrieval functions after calling endprofstr() .</p> <p>The functions getprofstrbyname() and getprofstr() are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function free_profstr() should be used to free the pointers returned by either getprofstrbyname() or getprofstr() .</p> <p>The parameter <i>name</i> must be a pointer to the profile name in the form of a null-terminated character-string.</p>

The parameter *src* may be set to any of `TSOL_SRC_FILES`, `TSOL_SRC_NISPLUS`, or `TSOL_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`. For most applications the *src* parameter should be set to `TSOL_SRC_SWITCH`, indicating that the system should use the `/etc/nsswitch.conf` file to determine the ultimate source of the database. However, in certain administrative application, it may be prudent to use the `TSOL_SRC_FILES` option to for a read from the `/etc/security/tsol/tsoluser` file or the `TSOL_SRC_NISPLUS` option to force a read from the `tsoluser` NIS+ database.

The function `putprofstr()` replaces an existing profile entry or adds a new entry if the profile name does not already exist. Currently the *src* parameter is treated as `TSOL_SRC_NISPLUS`, forcing all information to be written to the NIS+ table. Use of files or of `nsswitch.conf`(4) may be supported in future releases.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setprofstr()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getprofstr()`, the threads enumerate disjoint subsets of the `tsolprof`(4) database.

User entries are represented by the struct `profstr_t` structure defined in `<tsol/prof.h>`:

```
typedef struct profstr_t {
    char  name;      /* name of profile */
    char  desc;      /* description */
    char  auths;     /* comma separated list of authorization numbers */
    char  actions;   /* semicolon separated action descriptions */
    char  cmds;      /* semicolon separated command descriptions */
} profstr_t;
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The function `getprofstrbyname()` returns a pointer to a `profstr_t` if it successfully locates the requested entry; otherwise it returns `NULL`.

The function `getprofstr()` returns a pointer to a `profstr_t` if it successfully enumerates an entry; otherwise it returns `NULL`, indicating the end of the enumeration.

The function `putprofstr()` returns 0 on success.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

NOTES

Intro(3)

attributes(5)

Programs that use the interfaces described here cannot be linked statically since the implementation of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see *Intro(3)* , *Notes On Multithread Applications* , for information about the use of the `_REENTRANT` flag.

NAME	getsockopt, setsockopt – Get and set options on sockets						
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl [library...]</pre> <pre>#include <sys/types.h> #include <sys/socket.h> int getsockopt(int s, int level, int optname, void * optval, socklen_t * optlen); int setsockopt(int s, int level, int optname, const void * optval, socklen_t optlen);</pre>						
DESCRIPTION	<p>getsockopt() and setsockopt() manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost “socket” level.</p> <p>When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the “socket” level, <i>level</i> is specified as SOL_SOCKET . To manipulate options at any other level, <i>level</i> is the protocol number of the protocol that controls the option. For example, to indicate that an option is to be interpreted by the TCP protocol, <i>level</i> is set to the TCP protocol number (see getprotobyname(3N)).</p> <p>The parameters <i>optval</i> and <i>optlen</i> are used to access option values for setsockopt() . For getsockopt() , they identify a buffer in which the value(s) for the requested option(s) are to be returned. For getsockopt() , <i>optlen</i> is a value-result parameter, initially containing the size of the buffer pointed to by <i>optval</i> , and modified on return to indicate the actual size of the value returned. Use a 0 <i>optval</i> if no option value is to be supplied or returned.</p> <p><i>optname</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <sys/socket.h> contains definitions for the socket-level options described below. Options at other protocol levels vary in format and name.</p> <p>Most socket-level options take an int for <i>optval</i> . For setsockopt() , the <i>optval</i> parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a struct linger parameter that specifies the desired state of the option and the linger interval (see below). struct linger is defined in <sys/socket.h> . struct linger contains the following members:</p> <table> <tr> <td>l_onoff</td><td>on = 1/off = 0</td></tr> <tr> <td>l_linger</td><td>linger time, in seconds</td></tr> </table> <p>The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt() and set with setsockopt() .</p> <table> <tr> <td>SO_DEBUG</td><td>enable/disable recording of debugging information</td></tr> </table>	l_onoff	on = 1/off = 0	l_linger	linger time, in seconds	SO_DEBUG	enable/disable recording of debugging information
l_onoff	on = 1/off = 0						
l_linger	linger time, in seconds						
SO_DEBUG	enable/disable recording of debugging information						

SO_REUSEADDR	enable/disable local address reuse
SO_KEEPAIVE	enable/disable keep connections alive
SO_DONTROUTE	enable/disable routing bypass for outgoing messages
SO_LINGER	linger on close if data is present
SO_BROADCAST	enable/disable permission to transmit broadcast messages
SO_OOBINLINE	enable/disable reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_DGRAM_ERRIND	application wants delayed error
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules.

SO_REUSEADDR indicates that the rules used in validating addresses supplied in a `bind(3N)` call should allow reuse of local addresses. SO_KEEPAIVE enables the periodic transmission of messages on a connected socket. If the connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified using a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messages are queued on a socket and a `close(2)` is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the `close()` attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the `setsockopt()` call when SO_LINGER is requested). If SO_LINGER is disabled and a `close()` is issued, the system will process the `close()` in a manner that allows the process to continue as quickly as possible.

The option SO_BROADCAST requests permission to send broadcast datagrams on the socket. With protocols that support out-of-band data, the SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with `recv()` or `read()` calls without the MSG_OOB flag. No privilege is required to set the SO_BROADCAST flag, and any

user may do so; however, the `PRIV_NET_BROADCAST` privilege is required to use a broadcast address.

`SO_SNDBUF` and `SO_RCVBUF` are options that adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections or may be decreased to limit the possible backlog of incoming data. SunOS sets the maximum buffer size for both UDP and TCP to 256 Kbytes.

By default, delayed errors (such as ICMP port unreachable packets) are returned only for connected datagram sockets. `SO_DGRAM_ERRIND` makes it possible to receive errors for datagram sockets that are not connected. When this option is set, certain delayed errors received after completion of a `sendto()` or `sendmsg()` operation will cause a subsequent `sendto()` or `sendmsg()` operation using the same destination address (`to` parameter) to fail with the appropriate error. See `send(3N)`.

Finally, `SO_TYPE` and `SO_ERROR` are options used only with `getsockopt()`. `SO_TYPE` returns the type of the socket (for example, `SOCK_STREAM`). It is useful for servers that inherit sockets on startup. `SO_ERROR` returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

RETURN VALUES

`getsockopt()` returns:

0 On success.

-1 On failure, and sets `errno` to indicate the error.

ERRORS

The call succeeds unless:

<code>EBADF</code>	The argument <code>s</code> is not a valid file descriptor.
<code>ENOMEM</code>	There was insufficient memory available for the operation to complete.
<code>ENOPROTOOPT</code>	The option is unknown at the level indicated.
<code>ENOSR</code>	There were insufficient STREAMS resources available for the operation to complete.
<code>ENOTSOCK</code>	The argument <code>s</code> is not a socket.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

A process must have the `PRIV_NET_RAWACCESS` privilege in order to specify IP options 130 or 134 (`IPOPT_SEC` and `IPOPT_CIPSO` , respectively, as defined in `<inet/ip.h>`). The former refers to the Basic Security Option and the latter refers to the CIPSO option. A process must have the `PRIV_NET_BROADCAST` privilege to use a broadcast address.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`read(2)` , `bind(3N)`

**SunOS 5.7 Reference
Manual**

`close(2)` , `ioctl(2)` , `getprotobyname(3N)` , `recv(3N)` , `send(3N)` ,
`socket(3N)` , `attributes(5)`

NAME	getuserent, setuserent, enduserent, getuserentbyname, getuserentbyuid, free_userent – Get Trusted Solaris user security attributes
SYNOPSIS	<pre>cc [flag...] file... -ltsolddb -ltsol -lnsl -lcmd [library...] #include <tsol/user.h> userent_t *getuserentbyname(char *user, int src); userent_t *getuserentbyuid(uid_t uid, int src); userent_t *getuserent(int src); void setuserent(int stayopen, int src); void enduserent(int src); void free_userent(userent_t *userent);</pre>
DESCRIPTION	<p>These functions are used to obtain entries describing Trusted Solaris user attributes from the <code>tsoluser</code> NIS+ database.</p> <p><code>getuserentbyname()</code> searches for information for a user with the specified user name <i>user</i>. <code>getuserentbyuid()</code> searches for information for a user with the specified user ID <i>uid</i>.</p> <p>The functions <code>setuserent()</code>, <code>getuserent()</code>, and <code>enduserent()</code> are used to list user entries from the database. <code>setuserent()</code> sets (or resets) the enumeration to the beginning of the set of Trusted Solaris user entries. This function should be called before the first call to <code>getuserent()</code>. A call to <code>getuserbyname()</code> or <code>getuserentbyuid()</code> leaves the list position in an indeterminate state. If the <i>stayopen</i> flag is not zero, the system may keep allocated resources such as open file descriptors until a subsequent call to <code>enduserent()</code>.</p> <p>Successive calls to <code>getuserent()</code> return either successive entries or <code>NULL</code>, which indicates the end of the list.</p> <p><code>enduserent()</code> may be called when the caller expects to do no further user-entry-retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more user-entry-retrieval functions after calling <code>enduserent()</code>.</p> <p>The functions <code>getuserentbyname()</code>, <code>getuserentbyuid()</code>, and <code>getuserent()</code> are reentrant interfaces that allocate memory to store returned results, and are safe for use in both single-threaded and multithreaded applications. The function <code>free_userent()</code> is used to release memory allocated by these two functions. The parameter <i>user</i>, a pointer to the user name, must be a null-terminated character string. The parameter <i>uid</i> must be a <code>uid_t</code>.</p>

The parameter *src* may be set to `TSOL_DB_SRC_FILES`, `TSOL_DB_SRC_NISPLUS`, or `TSOL_DB_SRC_SWITCH`, which are defined in `<tsol/tsol.h>`.

For listing in multithreaded applications, the position within the list is a process-wide-property shared by all threads. `setuserent()` may be used in a multithreaded application but resets the list position for all threads. If multiple threads interleave calls to `getuserent()`, the threads will list disjoint subsets of the `tsoluser` database.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

User entries are represented by the struct `userent_t` structure defined in `<tsol/user.h>`:

```
typedef struct userent_s {
    char *name;           /* user associated with this entry */
    char *lock;           /* is account locked? */
    char *badlogins;       /* how many failed login attempts so far */
    char *generation;     /* method of password generation */
    char *profiles;       /* user profiles used */
    char *roles;          /* roles assumable */
    char *idletime;       /* minutes a workstation may remain idle */
    char *idlecmd;        /* what to do at when idletime reached */
    char *labelview;      /* can user see ADMIN_HI and ADMIN_LOW labels */
    char *labeltrans;     /* process security attributes for label translation */
    char *labelmin;       /* allowed low login labels */
    char *labelmax;       /* allowed high login labels */
    char *usertype;       /* normal, admin-role, or non-admin-role */
    char *res1;           /* reserved for future use */
    char *res2;           /* reserved for future use */
    char *res3;           /* reserved for future use */
    userent_t *next;
};
```

If it successfully locates the requested entry, `getuserentbyname()` returns a pointer to a `userent_t`. If unsuccessful, `getuserentbyname()` returns `NULL`.

If it successfully lists an entry, `getuserent()` returns a pointer to a struct `userent_t`. If unsuccessful, `getuserent()` returns `NULL`, indicating the end of the list.

Upon success, `setuserent()` and `enduserent()` return 0.

The functions `getuserentbyname()`, `getuserentbyuid()`, and `getuserent()` return `NULL` on failure.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`Intro(3)`

**SunOS 5.7 Reference
Manual**

`attributes(5)`

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically because the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see `Intro(3)`, `Notes On Multithread Applications`, for information about the use of the `_REENTRANT` flag.

These interfaces are uncommitted. Although they are not expected to change between minor releases of the Trusted Solaris environment, they may.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file);</pre>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a utmp structure with the following members:</p> <pre>char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutent() The <code>getutent()</code> function reads in the next entry from a utmp-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The <code>getutid()</code> function searches forward from the current point in the utmp file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The <code>getutline()</code> function searches forward from the current point in the utmp file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>

pututline() The `pututline()` function writes the supplied `utmp` structure into the `utmp` file. It uses `getutid()` to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of `pututline()` will have searched for the proper entry using one of the these functions. If so, `pututline()` will not search. If `pututline()` does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the `utmp` structure.

When called by a process that does not have an effective uid of 0 and a sensitivity label of `ADMIN_LOW`, `pututline()` invokes a program (that has the appropriate forced privileges) to verify and write the entry, since `/etc/utmpx` is normally writable only by a process with a UID of 0 and a sensitivity label of `ADMIN_LOW`. In this event, the `ut_name` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member must be a device special file and be writable by the user. If the process does not have the `PAF_TRUSTED_PATH` process attribute, all other fields in the entry are cleared.

setutent() The `setutent()` function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.

endutent() The `endutent()` function closes the currently open file.

utmpname() The `utmpname()` function allows the user to change the name of the file examined, from `/var/adm/utmp` to any other file. It is most often expected that this other file will be `/var/adm/wtmp`. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The `utmpname()` function does not open the file but closes the old file if it is currently open and saves the new file name.

RETURN VALUES A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, `utmpname()` returns 0. Otherwise, it returns 1.

USAGE These functions use buffered standard I/O for input, but `pututline()` uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the `utmp` and `wtmp` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on the `getutxent(3C)` manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES `/var/adm/utmp` User access and accounting information (old format)

<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

`pututline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututline()` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` functions, if the user has just modified those contents and passed the pointer back to `pututline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> ⇒ <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	type specified in <i>id</i> is <code>INIT_PROCESS</code> , <code>LOGIN_PROCESS</code> , <code>USER_PROCESS</code> , or <code>DEAD_PROCESS</code> , then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> ⇒ <code>ut_id</code> . If the end of file is reached without a match, it fails.
<code>getutxline()</code>	The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> ⇒ <code>ut_line</code> string. If the end of file is reached without a match, it fails.
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.
<code>endutxent()</code>	The <code>endutxent()</code> function closes the currently open file.
<code>utmpxname()</code>	The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code> . If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.
<code>getutmp()</code>	The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx()

The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp()

The `updwtmp()` function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx()

The `updwtmpx()` function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	stobl, stobcl, stobsl, stobil, stobclear – Translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int stobcl(const char * string, bclabel_t * label, const int flags, int * error); int stobsl(const char * string, bslabel_t * label, const int flags, int * error); int stobclear(const char * string, bclear_t * clearance, const int flags, int * error);</pre>				
DESCRIPTION	<p>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.</p> <p>The stobl 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.</p> <p>The generic form of an input character-coded label <i>string</i> is:</p> <pre>[+] [classification name] [[+ -] word ...]</pre> <p>Leading and trailing white space is ignored. Fields are separated by white space, a ' / ' (slash), or a ' , ' (comma). Case is irrelevant. If <i>string</i> starts with + or -, <i>string</i> is interpreted a modification to an existing label. If <i>string</i> starts with a classification name followed by a + or -, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. - modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels [see hextob()] when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> 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, <i>label</i> is validated as a defined label of the correct type dominated by the process's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>No corrections are made if there are errors in the character-coded label <i>string</i>. <i>string</i> must be complete and contain all the label components that are required by the label_encodings</td></tr> </table>	NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> must be complete and contain all the label components that are required by the label_encodings
NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> 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.

error is a return parameter that is set only if the function is unsuccessful.

stobcl() translates the character-coded CMW label string into a binary CMW label and places the result in the return parameter *label*. *string* has the form:

[*information label*] [[*information label*]]

or

'*'

Information Labels (IL s) are now obsolete. See NOTES below.

flags is the logical sum of NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL, or is 0 (zero). If both NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL are specified, the sensitivity label portion of *label* is set to the caller's present sensitivity label. The sensitivity label is translated first (unless ONLY_INFORMATION_LABEL is specified). Its presence is noted by the [character in the *string*. This translation must result in a sensitivity label that is dominated by the process's sensitivity label or an error is reported at the sensitivity label. The translated sensitivity label, or the one present in the *label* parameter must dominate the information label that is translated or an error is reported at the information label. Unless NO_CORRECTION is specified, these translations force the labels to dominate the minimum classification, and initial compartments set (and markings set) specified in the *label_encodings* file and correct the label to include other label components that are required by the *label_encodings* file, but not present in *string*. The special case where *string* contains '*' (star) sets the sensitivity label portion of *label* to the information level of the information label portion of *label*.

stobs1() translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *string* has the form: [[] *sensitivity label* []]

flags may 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 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*.

`stobil()` translates the character-coded information label string into a binary information label and places the result in the return parameter *label*. *string* has the form: *information label*. However, information labels (ILs) are now obsolete. See NOTES below.

flags may 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 and markings sets 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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *string* has the form: *clearance*.

flags may 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 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 may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These functions return:

- 1 If the translation was successful and a valid binary label was returned.
- 0 If an error occurred. `error` indicates the type of error.

ERRORS

If these functions returned zero, `error` contains one of these 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3),
blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3),
bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3),
labelvers(3), sbltos(3), label_encodings(4)

Trusted Solaris Developer's Guide, *Trusted Solaris user's document set*, and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

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 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 (and, if applicable, markings) 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.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.

- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.
- Options related to information labels in the `label_encodings(4)` file can be ignored:

```
Markings Name= Marks;  
Float Process Information Label;
```

NAME	stobl, stobcl, stobsl, stobil, stobclear – Translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>int stobcl(const char * string, bclabel_t * label, const int flags, int * error);</pre> <pre>int stobsl(const char * string, bslabel_t * label, const int flags, int * error);</pre> <pre>int stobclear(const char * string, bclear_t * clearance, const int flags, int * error);</pre>				
DESCRIPTION	<p>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.</p> <p>The stobl 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.</p> <p>The generic form of an input character-coded label <i>string</i> is:</p> <pre>[+] [classification name] [[+ -] word ...]</pre> <p>Leading and trailing white space is ignored. Fields are separated by white space, a ' / ' (slash), or a ' , ' (comma). Case is irrelevant. If <i>string</i> starts with + or – , <i>string</i> is interpreted a modification to an existing label. If <i>string</i> starts with a classification name followed by a + or – , the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i> . + modifies an existing label by adding words. – modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i> .</p> <p>The stobl functions also translate hexadecimal label representations into binary labels [see hextob()] when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i> .</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> 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, <i>label</i> is validated as a defined label of the correct type dominated by the process's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>No corrections are made if there are errors in the character-coded label <i>string</i> . <i>string</i> must be complete and contain all the label components that are required by the label_encodings</td></tr> </table>	NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> must be complete and contain all the label components that are required by the label_encodings
NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> 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.

error is a return parameter that is set only if the function is unsuccessful.

stobcl() translates the character-coded CMW label string into a binary CMW label and places the result in the return parameter *label*. *string* has the form:

[*information label*] [[*information label*]]

or

'*'

Information Labels (IL s) are now obsolete. See NOTES below.

flags is the logical sum of NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL, or is 0 (zero). If both NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL are specified, the sensitivity label portion of *label* is set to the caller's present sensitivity label. The sensitivity label is translated first (unless ONLY_INFORMATION_LABEL is specified). Its presence is noted by the [character in the *string*. This translation must result in a sensitivity label that is dominated by the process's sensitivity label or an error is reported at the sensitivity label. The translated sensitivity label, or the one present in the *label* parameter must dominate the information label that is translated or an error is reported at the information label. Unless NO_CORRECTION is specified, these translations force the labels to dominate the minimum classification, and initial compartments set (and markings set) specified in the *label_encodings* file and correct the label to include other label components that are required by the *label_encodings* file, but not present in *string*. The special case where *string* contains '*' (star) sets the sensitivity label portion of *label* to the information level of the information label portion of *label*.

stobsl() translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *string* has the form: [[] *sensitivity label* []]

flags may 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 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*.

`stobil()` translates the character-coded information label string into a binary information label and places the result in the return parameter *label*. *string* has the form: *information label*. However, information labels (ILs) are now obsolete. See NOTES below.

flags may 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 and markings sets 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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *string* has the form: *clearance*

flags may 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 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 may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These functions return:

- 1 If the translation was successful and a valid binary label was returned.
- 0 If an error occurred. `error` indicates the type of error.

ERRORS

If these functions returned zero, `error` contains one of these 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3),
blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3),
bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3),
labelvers(3), sbltos(3), label_encodings(4)

Trusted Solaris Developer's Guide, *Trusted Solaris user's document set*, and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

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 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 (and, if applicable, markings) 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.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.

- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.
- Options related to information labels in the `label_encodings(4)` file can be ignored:

```
Markings Name= Marks;  
Float Process Information Label;
```


NAME	stobl, stobcl, stobsl, stobil, stobclear – Translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int stobcl(const char * string, bclabel_t * label, const int flags, int * error); int stobsl(const char * string, bslabel_t * label, const int flags, int * error); int stobclear(const char * string, bclear_t * clearance, const int flags, int * error);</pre>				
DESCRIPTION	<p>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.</p> <p>The stobl 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.</p> <p>The generic form of an input character-coded label <i>string</i> is:</p> <pre>[+] [classification name] [[+ -] word ...]</pre> <p>Leading and trailing white space is ignored. Fields are separated by white space, a ' / ' (slash), or a ' , ' (comma). Case is irrelevant. If <i>string</i> starts with + or -, <i>string</i> is interpreted a modification to an existing label. If <i>string</i> starts with a classification name followed by a + or -, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. - modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels [see hextob()] when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> 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, <i>label</i> is validated as a defined label of the correct type dominated by the process's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>No corrections are made if there are errors in the character-coded label <i>string</i>. <i>string</i> must be complete and contain all the label components that are required by the label_encodings</td></tr> </table>	NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> must be complete and contain all the label components that are required by the label_encodings
NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> 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.

error is a return parameter that is set only if the function is unsuccessful.

`stobcl()` translates the character-coded CMW label string into a binary CMW label and places the result in the return parameter *label*. *string* has the form:

[*information label*] [[*information label*]]

or

'*'

Information Labels (IL s) are now obsolete. See NOTES below.

flags is the logical sum of NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL, or is 0 (zero). If both NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL are specified, the sensitivity label portion of *label* is set to the caller's present sensitivity label. The sensitivity label is translated first (unless ONLY_INFORMATION_LABEL is specified). Its presence is noted by the [character in the *string*. This translation must result in a sensitivity label that is dominated by the process's sensitivity label or an error is reported at the sensitivity label. The translated sensitivity label, or the one present in the *label* parameter must dominate the information label that is translated or an error is reported at the information label. Unless NO_CORRECTION is specified, these translations force the labels to dominate the minimum classification, and initial compartments set (and markings set) specified in the `label_encodings` file and correct the label to include other label components that are required by the `label_encodings` file, but not present in *string*. The special case where *string* contains '*' (star) sets the sensitivity label portion of *label* to the information level of the information label portion of *label*.

`stobs1()` translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *string* has the form: [[] *sensitivity label* []]

flags may 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 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*.

`stobil()` translates the character-coded information label string into a binary information label and places the result in the return parameter *label*. *string* has the form: *information label*. However, information labels (ILs) are now obsolete. See NOTES below.

flags may 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 and markings sets 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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *string* has the form: *clearance*.

flags may 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 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 may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These functions return:

- 1 If the translation was successful and a valid binary label was returned.
- 0 If an error occurred. `error` indicates the type of error.

ERRORS

If these functions returned zero, `error` contains one of these 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3),
blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3),
bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3),
labelvers(3), sbltos(3), label_encodings(4)

Trusted Solaris Developer's Guide, *Trusted Solaris user's document set*, and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

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 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 (and, if applicable, markings) 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.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.

- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.
- Options related to information labels in the `label_encodings(4)` file can be ignored:

```
Markings Name= Marks;  
Float Process Information Label;
```

NAME	stobl, stobcl, stobsl, stobil, stobclear – Translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...]</pre> <pre>#include <tsol/label.h></pre> <pre>int stobcl(const char * string, bclabel_t * label, const int flags, int * error);</pre> <pre>int stobsl(const char * string, bslabel_t * label, const int flags, int * error);</pre> <pre>int stobclear(const char * string, bclear_t * clearance, const int flags, int * error);</pre>				
DESCRIPTION	<p>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.</p> <p>The stobl 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.</p> <p>The generic form of an input character-coded label <i>string</i> is:</p> <pre>[+] [classification name] [[+ -] word ...]</pre> <p>Leading and trailing white space is ignored. Fields are separated by white space, a ' / ' (slash), or a ' , ' (comma). Case is irrelevant. If <i>string</i> starts with + or –, <i>string</i> is interpreted a modification to an existing label. If <i>string</i> starts with a classification name followed by a + or –, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. – modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels [see hextob()] when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> 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, <i>label</i> is validated as a defined label of the correct type dominated by the process's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>No corrections are made if there are errors in the character-coded label <i>string</i>. <i>string</i> must be complete and contain all the label components that are required by the label_encodings</td></tr> </table>	NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> must be complete and contain all the label components that are required by the label_encodings
NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> 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.

error is a return parameter that is set only if the function is unsuccessful.

stobl() translates the character-coded CMW label string into a binary CMW label and places the result in the return parameter *label*. *string* has the form:

[*information label*] [[*information label*]]

or

'*'

Information Labels (IL s) are now obsolete. See NOTES below.

flags is the logical sum of NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL, or is 0 (zero). If both NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL are specified, the sensitivity label portion of *label* is set to the caller's present sensitivity label. The sensitivity label is translated first (unless ONLY_INFORMATION_LABEL is specified). Its presence is noted by the [character in the *string*. This translation must result in a sensitivity label that is dominated by the process's sensitivity label or an error is reported at the sensitivity label. The translated sensitivity label, or the one present in the *label* parameter must dominate the information label that is translated or an error is reported at the information label. Unless NO_CORRECTION is specified, these translations force the labels to dominate the minimum classification, and initial compartments set (and markings set) specified in the *label_encodings* file and correct the label to include other label components that are required by the *label_encodings* file, but not present in *string*. The special case where *string* contains '*' (star) sets the sensitivity label portion of *label* to the information level of the information label portion of *label*.

stobl() translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *string* has the form: [[] *sensitivity label* []]

flags may 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 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*.

`stobil()` translates the character-coded information label string into a binary information label and places the result in the return parameter *label*. *string* has the form: *information label* However, information labels (ILs) are now obsolete. See NOTES below.

flags may 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 and markings sets 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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *string* has the form: *clearance*

flags may 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 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 may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These functions return:

- 1 If the translation was successful and a valid binary label was returned.
- 0 If an error occurred. `error` indicates the type of error.

ERRORS

If these functions returned zero, `error` contains one of these 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3) , bilconjoin(3) , blcompare(3) , blinset(3) ,
blmanifest(3) , blminmax(3) , blportion(3) , bltocolour(3) , bltos(3)
, bltype(3) , blvalid(3) , btohex(3) , hextob(3) , labelinfo(3) ,
labelvers(3) , sbltos(3) , label_encodings(4)

Trusted Solaris Developer's Guide , *Trusted Solaris user's document set* , and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

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 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 (and, if applicable, markings) 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.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.

- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.
- Options related to information labels in the `label_encodings(4)` file can be ignored:

```
Markings Name= Marks;  
Float Process Information Label;
```

NAME	stobl, stobcl, stobsl, stobil, stobclear – Translate character-coded labels to binary labels				
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int stobcl(const char * string, bclabel_t * label, const int flags, int * error); int stobsl(const char * string, bslabel_t * label, const int flags, int * error); int stobclear(const char * string, bclear_t * clearance, const int flags, int * error);</pre>				
DESCRIPTION	<p>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.</p> <p>The stobl 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.</p> <p>The generic form of an input character-coded label <i>string</i> is:</p> <pre>[+] [classification name] [[+ -] word ...]</pre> <p>Leading and trailing white space is ignored. Fields are separated by white space, a ' / ' (slash), or a ' , ' (comma). Case is irrelevant. If <i>string</i> starts with + or -, <i>string</i> is interpreted a modification to an existing label. If <i>string</i> starts with a classification name followed by a + or -, the new classification is used and the rest of the old label is retained and modified as specified by <i>string</i>. + modifies an existing label by adding words. - modifies an existing label by removing words. To the maximum extent possible, errors in <i>string</i> are corrected in the resulting binary label <i>label</i>.</p> <p>The stobl functions also translate hexadecimal label representations into binary labels [see hextob()] when the string starts with 0x and either NEW_LABEL or NO_CORRECTION is specified in <i>flags</i>.</p> <p><i>flags</i> may be the following:</p> <table> <tr> <td>NEW_LABEL</td><td><i>label</i> 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, <i>label</i> is validated as a defined label of the correct type dominated by the process's sensitivity label.</td></tr> <tr> <td>NO_CORRECTION</td><td>No corrections are made if there are errors in the character-coded label <i>string</i>. <i>string</i> must be complete and contain all the label components that are required by the label_encodings</td></tr> </table>	NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> must be complete and contain all the label components that are required by the label_encodings
NEW_LABEL	<i>label</i> 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, <i>label</i> 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 <i>string</i> . <i>string</i> 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.

error is a return parameter that is set only if the function is unsuccessful.

stobcl() translates the character-coded CMW label string into a binary CMW label and places the result in the return parameter *label*. *string* has the form:

[*information label*] [[*information label*]]

or

'*'

Information Labels (IL s) are now obsolete. See NOTES below.

flags is the logical sum of NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL, or is 0 (zero). If both NEW_LABEL or NO_CORRECTION and ONLY_INFORMATION_LABEL are specified, the sensitivity label portion of *label* is set to the caller's present sensitivity label. The sensitivity label is translated first (unless ONLY_INFORMATION_LABEL is specified). Its presence is noted by the [character in the *string*. This translation must result in a sensitivity label that is dominated by the process's sensitivity label or an error is reported at the sensitivity label. The translated sensitivity label, or the one present in the *label* parameter must dominate the information label that is translated or an error is reported at the information label. Unless NO_CORRECTION is specified, these translations force the labels to dominate the minimum classification, and initial compartments set (and markings set) specified in the *label_encodings* file and correct the label to include other label components that are required by the *label_encodings* file, but not present in *string*. The special case where *string* contains '*' (star) sets the sensitivity label portion of *label* to the information level of the information label portion of *label*.

stobsl() translates the character-coded sensitivity label string into a binary sensitivity label and places the result in the return parameter *label*. *string* has the form: [[] *sensitivity label* []]

flags may 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 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*.

`stobil()` translates the character-coded information label string into a binary information label and places the result in the return parameter *label*. *string* has the form: *information label*. However, information labels (ILs) are now obsolete. See NOTES below.

flags may 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 and markings sets 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*.

`stobclear()` translates the character-coded clearance string into a binary clearance and places the result in the return parameter *clearance*. *string* has the form: *clearance*.

flags may 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 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 may not be the same as the translation of a sensitivity label. These functions use different tables of the `label_encodings` file that may contain different words and constraints.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These functions return:

- 1 If the translation was successful and a valid binary label was returned.
- 0 If an error occurred. `error` indicates the type of error.

ERRORS

If these functions returned zero, `error` contains one of these 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

bcltobanner(3), bilconjoin(3), blcompare(3), blinset(3),
blmanifest(3), blminmax(3), blportion(3), bltocolour(3), bltos(3),
bltype(3), blvalid(3), btohex(3), hextob(3), labelinfo(3),
labelvers(3), sbltos(3), label_encodings(4)

Trusted Solaris Developer's Guide, *Trusted Solaris user's document set*, and
Trusted Solaris administrator's document set

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

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 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 (and, if applicable, markings) 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.

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW.

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL]; however, the IL component is fixed at ADMIN_LOW.

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.

- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW` .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always `ADMIN_LOW` , and cannot be set on any objects.
- Options related to information labels in the `label_encodings(4)` file can be ignored:

```
Markings Name= Marks;  
Float Process Information Label;
```

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – Translate and verify user authorizations
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id); char auth_set_to_str(auth_set_t * authset, char separator); auth_t str_to_auth(char * auth_name); auth_set_t *str_to_auth_set(char * auth_names, char * separators); char *get_auth_text(auth_t auth_id); int chkauth(auth_t auth_id, char * username); void free_auth_set(auth_set_t * authset);</pre>
DESCRIPTION	<p>auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i> . If <i>auth_id</i> is an undefined authorization ID , the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID , TSOL_MAX_AUTH , a NULL is returned.</p> <p>auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C) . Authorization names are separated by the character <i>separator</i> . Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list.</p> <p>str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i> . <i>auth_name</i> is the name in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH .</p> <p>str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i> . <i>auth_names</i> are names in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . The string none is translated to an empty authorization list (NULL) . This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization</p>

names appear in the *auth_names* string, the function converts these invalid authorizations into *auth_id*s with the value of 0 .

get_auth_text() returns a pointer to the statically-allocated, null-terminated authorization description text specified by *auth_id* .

chkauth() returns 1 if the user, specified by *username* has the authorization specified by *auth_id* .

free_auth_set() releases the memory returned by *str_to_auth_set*() .

RETURN VALUES

auth_to_str() returns a pointer to the translated authorization name string. It returns NULL on failure.

str_to_auth() returns the numeric authorization id. It returns 0 on failure.

auth_set_to_str() returns a pointer to the translated authorization names string. It returns NULL on failure.

str_to_auth_set() returns NULL on success.

get_auth_text() return a string on success and NULL upon failure.

chkauth() returns 1 on success and 0 on failure.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

NOTES

This interface is uncommitted, which means it may change between minor releases of the Trusted Solaris environment. To use these routines, the program must be loaded with the Trusted Solaris library *libtsolddb* .

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auth_desc(4) , *auth_name*(4)

attributes(5)

NAME	auth_to_str, str_to_auth, auth_set_to_str, str_to_auth_set, free_auth_set, get_auth_text, chkauth – Translate and verify user authorizations
SYNOPSIS	<pre>cc [flag...] file... -ltsol -ltsolddb -lcmd -lnsl [library...] #include <tsol/auth.h> char *auth_to_str(auth_t auth_id); char auth_set_to_str(auth_set_t * authset, char separator); auth_t str_to_auth(char * auth_name); auth_set_t *str_to_auth_set(char * auth_names, char * separators); char *get_auth_text(auth_t auth_id); int chkauth(auth_t auth_id, char * username); void free_auth_set(auth_set_t * authset);</pre>
DESCRIPTION	<p>auth_to_str() returns a pointer to the statically allocated, null-terminated text authorization name specified by <i>auth_id</i> . If <i>auth_id</i> is an undefined authorization ID , the integer ordinal of <i>auth_id</i> is returned. If <i>auth_id</i> is greater than the maximum allowable authorization ID , TSOL_MAX_AUTH , a NULL is returned.</p> <p>auth_set_to_str() places the name of each authorization in <i>authset</i> into a string which is allocated and returned by the function. The memory for this string may be released with free(3C) . Authorization names are separated by the character <i>separator</i> . Integer ordinals are used to name the undefined authorizations found in the authorization list. The string none is returned when representing an empty authorization list.</p> <p>str_to_auth() returns the numeric authorization ID specified by the null-terminated text authorization name <i>auth_name</i> . <i>auth_name</i> is the name in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . Authorization names can be specified in upper or lower case. An integer ordinal in the string is also acceptable. This function returns 0 when the string cannot be translated, or when an integer ordinal in the string is greater than TSOL_MAX_AUTH .</p> <p>str_to_auth_set() breaks the <i>auth_names</i> string into tokens to be translated into an authorization list based on the token separators <i>separators</i> . <i>auth_names</i> are names in the Names field of auth_name(4) ; this function does not parse the names in the Manifest Constant field of auth_name(4) . The string none is translated to an empty authorization list (NULL) . This function returns a pointer to an <i>auth_set_t</i> on success or a NULL if either the <i>auth_names</i> parameter is NULL or the function cannot allocate enough memory. If some invalid authorization</p>

names appear in the *auth_names* string, the function converts these invalid authorizations into *auth_id*s with the value of 0 .

get_auth_text() returns a pointer to the statically-allocated, null-terminated authorization description text specified by *auth_id* .

chkauth() returns 1 if the user, specified by *username* has the authorization specified by *auth_id* .

free_auth_set() releases the memory returned by *str_to_auth_set*() .

auth_to_str() returns a pointer to the translated authorization name string. It returns NULL on failure.

str_to_auth() returns the numeric authorization id. It returns 0 on failure.

auth_set_to_str() returns a pointer to the translated authorization names string. It returns NULL on failure.

str_to_auth_set() returns NULL on success.

get_auth_text() return a string on success and NULL upon failure.

chkauth() returns 1 on success and 0 on failure.

RETURN VALUES

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

NOTES

This interface is uncommitted, which means it may change between minor releases of the Trusted Solaris environment. To use these routines, the program must be loaded with the Trusted Solaris library *libtsolddb* .

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

auth_desc(4) , *auth_name*(4)

attributes(5)

NAME priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number

SYNOPSIS cc [*flag...*] *file...* -ltsol [*library...*]

```
#include <tsol/priv.h>
priv_t str_to_priv(const char * priv_name);

char * priv_to_str(const priv_t priv_id);

char * str_to_priv_set(const char * priv_names, priv_set_t * priv_set, const char *
separators);

char * priv_set_to_str(priv_set_t * priv_set, char separator, char * buffer, int * buflen);

char * get_priv_text(const priv_t priv_id);
```

DESCRIPTION

priv_to_str() returns a pointer to the statically allocated, null-terminated privilege name specified by *priv_id*. If *priv_id* is an undefined privilege ID , the integer ordinal of *priv_id* is returned. If *priv_id* is greater than TSOL_MAX_PRIV , the maximum allowable privilege ID , a NULL is returned.

str_to_priv() returns the numeric privilege ID specified by the null-terminated privilege name *priv_name*. Privilege names can be specified in upper or lower case. An integer ordinal in the string is also acceptable.

priv_set_to_str() appends the name of each privilege in *priv_set* to a string to which the user-supplied *buffer* of length *buflen* points. Privilege names are separated by the *separator* character. Integer ordinals name the undefined privileges found in the privilege set. String *none* identifies an empty privilege set; and *all* , a full privilege set. Privilege names in the string are sorted in alphabetical order by localized sort.

Based on the token separators (*separators*), str_to_priv_set() breaks the *priv_names* string into tokens to be translated into a privilege set. Token *none* is translated to an empty privilege set; token *all* , to a full privilege set. The presence of token *none* overrides whatever precedes it. For example, the string *file_mac_read,file_mac_write,none,proc_nofloat* produces the same result as *proc_nofloat* alone. The constructed privilege set is stored in the *priv_set_t* buffer to which *priv_set* points.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

<code>get_priv_text()</code>	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
<code>priv_to_str()</code>	Returns a pointer to the translated privilege name string. The function returns <code>NULL</code> and sets <code>errno</code> on failure.
<code>str_to_priv()</code>	Returns the numeric privilege ID. The function returns <code>-1</code> and sets <code>errno</code> on failure.
<code>priv_set_to_str()</code>	Returns a pointer to the translated privilege names string. If the passed-in <i>buflen</i> is too small to hold the string, this routine stores the required buffer size into <i>buflen</i> and returns <code>NULL</code> . The function returns <code>NULL</code> and sets <code>errno</code> on failure. This function returns <code>-1</code> if the string cannot be translated or if an integer ordinal in the string is greater than <code>TSOL_MAX_PRIV</code> .
<code>str_to_priv_set()</code>	Returns <code>NULL</code> on success. If bad privilege names appear in the <i>priv_names</i> string, the function returns a pointer to the first privilege name that is not recognizable.

ERRORS

<code>priv_to_str()</code> may fail for this reason:	
<code>EINVAL</code>	The specified <i>priv_id</i> is greater than <code>TSOL_MAX_PRIV</code> .
<code>priv_set_to_str()</code> may fail for this reason:	
<code>EFAULT</code>	The specified <i>priv_set</i> is an invalid address.
<code>str_to_priv()</code> may fail for one of these reasons:	
<code>EINVAL</code>	The specified <i>priv_name</i> does not match any of the defined privilege names.
<code>EFAULT</code>	The specified <i>priv_name</i> is an invalid address.

NOTES

To use these routines, the program must be loaded with the Trusted Solaris library `libtsol` or `libtsol.so`.

SEE ALSO

Trusted Solaris 7
Reference Manual

`priv_desc(4)` `priv_name(4)`

NAME | priv_to_str, priv_set_to_str, str_to_priv, str_to_priv_set, get_priv_text – Convert a numeric privilege to its name or a privilege name to its number

SYNOPSIS | cc [flag...] file... -ltsol [library...]

```
#include <tsol/priv.h>
priv_t str_to_priv(const char * priv_name);

char * priv_to_str(const priv_t priv_id);

char * str_to_priv_set(const char * priv_names, priv_set_t * priv_set, const char *
separators);

char * priv_set_to_str(priv_set_t * priv_set, char separator, char * buffer, int * buflen);

char * get_priv_text(const priv_t priv_id);
```

DESCRIPTION

priv_to_str() returns a pointer to the statically allocated, null-terminated privilege name specified by *priv_id*. If *priv_id* is an undefined privilege ID , the integer ordinal of *priv_id* is returned. If *priv_id* is greater than TSOL_MAX_PRIV , the maximum allowable privilege ID , a NULL is returned.

str_to_priv() returns the numeric privilege ID specified by the null-terminated privilege name *priv_name*. Privilege names can be specified in upper or lower case. An integer ordinal in the string is also acceptable.

priv_set_to_str() appends the name of each privilege in *priv_set* to a string to which the user-supplied *buffer* of length *buflen* points. Privilege names are separated by the *separator* character. Integer ordinals name the undefined privileges found in the privilege set. String *none* identifies an empty privilege set; and *all* , a full privilege set. Privilege names in the string are sorted in alphabetical order by localized sort.

Based on the token separators (*separators*), str_to_priv_set() breaks the *priv_names* string into tokens to be translated into a privilege set. Token *none* is translated to an empty privilege set; token *all* , to a full privilege set. The presence of token *none* overrides whatever precedes it. For example, the string *file_mac_read,file_mac_write,none,proc_nofloat* produces the same result as *proc_nofloat* alone. The constructed privilege set is stored in the *priv_set_t* buffer to which *priv_set* points.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

<code>get_priv_text()</code>	Returns a pointer to the statically allocated, null-terminated privilege description text specified by <i>priv_id</i> .
<code>priv_to_str()</code>	Returns a pointer to the translated privilege name string. The function returns <code>NULL</code> and sets <code>errno</code> on failure.
<code>str_to_priv()</code>	Returns the numeric privilege ID. The function returns <code>-1</code> and sets <code>errno</code> on failure.
<code>priv_set_to_str()</code>	Returns a pointer to the translated privilege names string. If the passed-in <i>buflen</i> is too small to hold the string, this routine stores the required buffer size into <i>buflen</i> and returns <code>NULL</code> . The function returns <code>NULL</code> and sets <code>errno</code> on failure. This function returns <code>-1</code> if the string cannot be translated or if an integer ordinal in the string is greater than <code>TSOL_MAX_PRIV</code> .
<code>str_to_priv_set()</code>	Returns <code>NULL</code> on success. If bad privilege names appear in the <i>priv_names</i> string, the function returns a pointer to the first privilege name that is not recognizable.

ERRORS

<code>priv_to_str()</code> may fail for this reason:	
<code>EINVAL</code>	The specified <i>priv_id</i> is greater than <code>TSOL_MAX_PRIV</code> .
<code>priv_set_to_str()</code> may fail for this reason:	
<code>EFAULT</code>	The specified <i>priv_set</i> is an invalid address.
<code>str_to_priv()</code> may fail for one of these reasons:	
<code>EINVAL</code>	The specified <i>priv_name</i> does not match any of the defined privilege names.
<code>EFAULT</code>	The specified <i>priv_name</i> is an invalid address.

NOTES

To use these routines, the program must be loaded with the Trusted Solaris library `libtsol` or `libtsol.so`.

SEE ALSO

Trusted Solaris 7
Reference Manual

`priv_desc(4)` `priv_name(4)`

**SunOS 5.7 Reference
Manual**

attributes(5)

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
DESCRIPTION	<p>These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code>], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (* procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char * nettype);</code></p> <p>Register program <i>prognum</i>, procedure <i>procname</i>, and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i>, version <i>versnum</i>, and procedure <i>procnum</i>, <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its static result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters while <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i>. See <code>rpc(3N)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p> <p><code>int svc_reg(const SVCXPRT * xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (* dispatch)(), const struct netconfig * netconf);</code></p> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i> \Rightarrow <i>nc_netid</i>] to <i>xpirt</i> \Rightarrow <i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p>

```
void svc_unreg(const rpcprog_t prognum , const rpcvers_t versnum );
```

Remove from the `rpcbind` service, all mappings of the triple [*prognum* , *versnum* , *all-transport*] to network address and all mappings within the RPC service package of the double [*prognum* , *versnum*] to dispatch routines.

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

The `PRIV_NET_SETID` privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

```
int svc_auth_reg(const int cred_flavor , const enum auth_stat (*handler)());
```

Registers the service authentication routine *handler* with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type *cred_flavor* . This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call `svc_auth_reg()` after registering the service and prior to calling `svc_run()` . When needed to process an RPC credential of type *cred_flavor* , the *handler* procedure will be called with two parameters (`struct svc_req * rqst` , `struct rpc_msg * msg`) and is expected to return a valid `enum auth_stat` value. There is no provision to change or delete an authentication handler once registered.

The `svc_auth_reg()` routine returns 0 if the registration is successful, 1 if *cred_flavor* already has an authentication handler registered for it, and -1 otherwise.

```
void xpirt_register(const SVCXPRT * xprt );
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3N)`). Service implementors usually do not need this routine.

```
void xpirt_unregister(const SVCXPRT * xprt );
```

Before an RPC service transport handle *xprt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable `svc_fdset` [see `rpc_svc_calls(3N)`]. Service implementors usually do not need this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is on when `rpc_reg()` or `rpc_svc()` is called, a multilevel mapping is created. To delete a multilevel mapping, `svc_unreg()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The `PRIV_NET_SETID` privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`,
`rpc_svc_create(3N)`, `rpcbind(3N)`

**SunOS 5.7 Reference
Manual**

`select(3C)`, `rpc_svc_err(3N)`, `attributes(5)`

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT *svc, const uint_t req, void *info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*

int svc_create(const void (dispatch)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);*
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

*void svc_destroy(SVCXPRT * *xprt*);*

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

*SVCXPRT *svc_dg_create(const int *fildez*, const uint_t *sendsz*, const uint_t *recvsz*);*

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *rcvsvz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildev* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildev*, const uint_t *sendsz*, const uint_t *rcvsvz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *rcvsvz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns *NULL* if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see *clnt_raw_create()* in *rpc_clnt_create(3N)*). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns *NULL* if it fails, and an error message is logged.

Note: *svc_run()* should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildev*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *rcvsvz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildev* is the file descriptor on which the service is listening. If *fildev* is *RPC_ANYFD*, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildev* is bound to the address specified by *bindaddr*, otherwise *fildev* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *rcvsvz*; values of 0 choose suitable defaults. This routine returns *NULL* if it fails, and an error message is logged. The server is not registered with the *rpcbind(1M)* service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *progrnum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

svc_tp_create() creates a server handle for the network specified by *netconf*, and registers itself with the *rpcbind* service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a NULL is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev , const uint_t sendsz , const uint_t recvsz);`
 This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to NULL . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N), attributes(5)

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles		
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>		
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT * svc, const uint_t req, void * info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p>		
	<table> <tr> <td data-bbox="381 567 730 1134">SVCGET_VERSQUIET</td><td data-bbox="730 567 1302 1134"> <p>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</p> </td></tr> </table>	SVCGET_VERSQUIET	<p>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</p>
SVCGET_VERSQUIET	<p>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</p>		
	<table> <tr> <td data-bbox="381 1155 730 1499">SVCSET_VERSQUIET</td><td data-bbox="730 1155 1302 1499"> <p>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</p> </td></tr> </table>	SVCSET_VERSQUIET	<p>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</p>
SVCSET_VERSQUIET	<p>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</p>		

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*.

int *svc_create*(const void (* *dispatch*)(const struct *svc_req* *, const SVCXPRT *), const *rpcprog_t* *prognum*, const *rpcvers_t* *versnum*, const char * *nettype*);
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in *NETPATH* variable or in top to bottom order in the *netconfig* database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

void *svc_destroy*(SVCXPRT * *xprt*);

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

SVCXPRT **svc_dg_create*(const *int* *fildez*, const *uint_t* *sendsz*, const *uint_t* *recvsz*);

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildes* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildes*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns *NULL* if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see *clnt_raw_create()* in *rpc_clnt_create(3N)*). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns *NULL* if it fails, and an error message is logged.

Note: *svc_run()* should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildes*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildes* is the file descriptor on which the service is listening. If *fildes* is *RPC_ANYFD*, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildes* is bound to the address specified by *bindaddr*, otherwise *fildes* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns *NULL* if it fails, and an error message is logged. The server is not registered with the *rpcbind(1M)* service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

svc_tp_create() creates a server handle for the network specified by *netconf*, and registers itself with the *rpcbind* service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev, const uint_t sendsz, const uint_t recvsz);`
 This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N) , attributes(5)

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT *svc, const uint_t req, void *info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create*(3N)

```
int svc_create(const void (* dispatch)(const struct svc_req *, const SVCXPRT *),
const rpcprog_t prognum, const rpcvers_t versnum, const char * nettype);
    svc_create() creates server handles for all the transports belonging
    to the class nettype.
```

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind*(1M)]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg*(3N)). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

```
void svc_destroy(SVCXPRT * xprt);
```

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

```
SVCXPRT *svc_dg_create(const int fildes, const uint_t sendsz, const uint_t
recvsz);
```

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildev* should be open and bound. The server is not registered with `rpcbind(1M)`.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildev*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns `NULL` if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see `clnt_raw_create()` in `rpc_clnt_create(3N)`). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns `NULL` if it fails, and an error message is logged.

Note: `svc_run()` should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildev*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildev* is the file descriptor on which the service is listening. If *fildev* is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildev* is bound to the address specified by *bindaddr*, otherwise *fildev* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

`svc_tp_create()` creates a server handle for the network specified by *netconf*, and registers itself with the `rpcbind` service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a NULL is returned and an error message is logged.

SVCXPRT *svc_vc_create(const int *fildev* , const uint_t *sendsz* , const uint_t *rcvsvsz*);

This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns NULL if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to NULL . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO Trusted Solaris 7 Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N), attributes(5)

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT * svc, const uint_t req, void * info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*.

int *svc_create*(const void (* *dispatch*)(const struct *svc_req* *, const SVCXPRT *), const *rpcprog_t* *prognum*, const *rpcvers_t* *versnum*, const char * *nettype*);
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in *NETPATH* variable or in top to bottom order in the *netconfig* database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

void *svc_destroy*(SVCXPRT * *xprt*);

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

SVCXPRT **svc_dg_create*(const *int* *fildes*, const *uint_t* *sendsz*, const *uint_t* *recvsz*);

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildes* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildes*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns *NULL* if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see *clnt_raw_create()* in *rpc_clnt_create(3N)*). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns *NULL* if it fails, and an error message is logged.

Note: *svc_run()* should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildes*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildes* is the file descriptor on which the service is listening. If *fildes* is *RPC_ANYFD*, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildes* is bound to the address specified by *bindaddr*, otherwise *fildes* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns *NULL* if it fails, and an error message is logged. The server is not registered with the *rpcbind(1M)* service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

svc_tp_create() creates a server handle for the network specified by *netconf*, and registers itself with the *rpcbind* service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev, const uint_t sendsz, const uint_t recvsz);`
 This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N) , attributes(5)

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See <code>NOTES</code> below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the `User` MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of `pollfd` structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT * svc, const uint_t req, void * info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*.

int *svc_create*(const void (* *dispatch*)(const struct *svc_req* *, const SVCXPRT *), const *rpcprog_t* *prognum*, const *rpcvers_t* *versnum*, const char * *nettype*);
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in *NETPATH* variable or in top to bottom order in the *netconfig* database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

void *svc_destroy*(SVCXPRT * *xprt*);

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

SVCXPRT **svc_dg_create*(const *int* *fildez*, const *uint_t* *sendsz*, const *uint_t* *recvsz*);

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildes* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildes*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns *NULL* if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see *clnt_raw_create()* in *rpc_clnt_create(3N)*). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns *NULL* if it fails, and an error message is logged.

Note: *svc_run()* should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildes*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildes* is the file descriptor on which the service is listening. If *fildes* is *RPC_ANYFD*, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildes* is bound to the address specified by *bindaddr*, otherwise *fildes* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns *NULL* if it fails, and an error message is logged. The server is not registered with the *rpcbind(1M)* service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

svc_tp_create() creates a server handle for the network specified by *netconf*, and registers itself with the *rpcbind* service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev, const uint_t sendsz, const uint_t recvsz);`
 This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N) , attributes(5)

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the `User` MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpcaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

SunOS 5.7 Reference
Manual

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the `User` MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreqset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See <code>NOTES</code> below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the `User` MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreqset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpcaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the `User` MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpcaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See <code>NOTES</code> below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT * svc, const uint_t req, void * info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*.

int *svc_create*(const void (* *dispatch*)(const struct *svc_req* *, const SVCXPRT *), const *rpcprog_t* *prognum*, const *rpcvers_t* *versnum*, const char * *nettype*);
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in *NETPATH* variable or in top to bottom order in the *netconfig* database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

void *svc_destroy*(SVCXPRT * *xprt*);

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

SVCXPRT **svc_dg_create*(const *int* *fildez*, const *uint_t* *sendsz*, const *uint_t* *recvsz*);

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildes* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildes*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns *NULL* if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see *clnt_raw_create()* in *rpc_clnt_create(3N)*). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns *NULL* if it fails, and an error message is logged.

Note: *svc_run()* should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildes*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildes* is the file descriptor on which the service is listening. If *fildes* is *RPC_ANYFD*, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildes* is bound to the address specified by *bindaddr*, otherwise *fildes* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns *NULL* if it fails, and an error message is logged. The server is not registered with the *rpcbind(1M)* service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

svc_tp_create() creates a server handle for the network specified by *netconf*, and registers itself with the *rpcbind* service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev, const uint_t sendsz, const uint_t recvsz);`
 This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N) , attributes(5)

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xprt_register, xprt_unregister – Library routines for registering servers
DESCRIPTION	<p>These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code>], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t</code> <code>rpc_reg(const rpcprog_t prognum , const rpcvers_t versnum , const rpcproc_t procnum , char * (* procname)(), const xdrproc_t inproc , const xdrproc_t outproc , const char * nettype);</code></p> <p>Register program <i>prognum</i> , procedure <i>procname</i> , and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i> , version <i>versnum</i> , and procedure <i>procnum</i> , <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its <code>static</code> result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters while <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i> . See <code>rpc(3N)</code> . This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p> <p><code>int</code> <code>svc_reg(const SVCXPRT * xprt , const rpcprog_t prognum , const rpcvers_t versnum , const void (* dispatch)(), const struct netconfig * netconf);</code></p> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i> . If <i>netconf</i> is <code>NULL</code> , the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i> , <i>versnum</i> , <i>netconf</i> \Rightarrow <i>nc_netid</i>] to <i>xprt</i> \Rightarrow <i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p>

```
void svc_unreg(const rpcprog_t prognum , const rpcvers_t versnum );
```

Remove from the `rpcbind` service, all mappings of the triple [*prognum* , *versnum* , *all-transports*] to network address and all mappings within the RPC service package of the double [*prognum* , *versnum*] to dispatch routines.

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

The `PRIV_NET_SETID` privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

```
int svc_auth_reg(const int cred_flavor , const enum auth_stat (*handler)());
```

Registers the service authentication routine *handler* with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type *cred_flavor* . This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call `svc_auth_reg()` after registering the service and prior to calling `svc_run()` . When needed to process an RPC credential of type *cred_flavor* , the *handler* procedure will be called with two parameters (`struct svc_req * rqst` , `struct rpc_msg * msg`) and is expected to return a valid `enum auth_stat` value. There is no provision to change or delete an authentication handler once registered.

The `svc_auth_reg()` routine returns 0 if the registration is successful, 1 if *cred_flavor* already has an authentication handler registered for it, and -1 otherwise.

```
void xprt_register(const SVCXPRT * xprt );
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3N)`). Service implementors usually do not need this routine.

```
void xprt_unregister(const SVCXPRT * xprt );
```

Before an RPC service transport handle *xprt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable `svc_fdset` [see `rpc_svc_calls(3N)`]. Service implementors usually do not need this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is on when `rpc_reg()` or `rpc_svc()` is called, a multilevel mapping is created. To delete a multilevel mapping, `svc_unreg()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The `PRIV_NET_SETID` privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`,
`rpc_svc_create(3N)`, `rpcbind(3N)`

`select(3C)`, `rpc_svc_err(3N)`, `attributes(5)`

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the `User` MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the `Automatic` MT mode or the `User` MT mode should never be read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfdp , const int pollretval) ;`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfdp* is the array of *pollfd* structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpccaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpcaller, svc_max_pollfd, svc_pollfd, svc_run, svc_sendreply – Library routines for RPC servers
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.</p> <p>These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as <code>svc_run()</code>) are called when the server is initiated.</p> <p>In the current implementation, the service transport handle <code>SVCXPRT</code> contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the <code>Automatic</code> or <code>User MT</code> modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.</p> <p>Programs can retrieve network security attributes from incoming requests. Privileged programs can create multilevel ports, create multilevel mappings, and set the security attributes of outgoing replies. See <code>SUMMARY OF TRUSTED SOLARIS CHANGES</code> for more information.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>int svc_dg_enablecache(SVCXPRT * <i>xprt</i> , const uint_t <i>cache_size</i>);</code> This function allocates a duplicate request cache for the service endpoint <i>xprt</i> , large enough to hold <i>cache_size</i> entries. Once enabled, there is no way to disable caching. This routine returns 1 if space necessary for a cache of the given size was successfully allocated, and 0 otherwise.</p> <p>This function is safe in MT applications.</p> <p><code>int svc_done(SVCXPRT * <i>xprt</i>);</code> This function frees resources allocated to service a client request directed to the service endpoint <i>xprt</i> . This call pertains only to servers executing in the <code>User MT</code> mode. In the <code>User MT</code> mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After <code>svc_done()</code> is invoked, the service endpoint <i>xprt</i> should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the <code>rpc_control()</code> call.</p>

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

`void svc_exit(void);`

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes `svc_run()` to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the `rpc_svc_create(3N)` functions, or using `xprt_register(3N)`.

`svc_exit()` has global scope and ends all RPC server activity.

`fd_set svc_fdset;`

A global variable reflecting the RPC server's read file descriptor bit mask.

This is only of interest if service implementors do not call `svc_run()`, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getregset()` or any creation routines. Do not pass its address to `select(3C)`! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`svc_fdset` is limited to 1024 file descriptors and is considered obsolete. Use of `svc_pollfd` is recommended instead.

`pollfd_t *svc_pollfd;`

A global variable pointing to an array of `pollfd_t` structures reflecting the RPC server's read file descriptor array. This is only of interest if service implementors do not call `svc_run()` but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines. Do not pass its address to `poll(2)`! Instead, pass the address of a copy.

By default, `svc_pollfd` is limited to 1024 entries. Use `rpc_control(3N)` to remove this limitation.

MT applications executing in either the Automatic MT mode or the User MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

`int svc_max_pollfd;`

A global variable containing the maximum length of the `svc_pollfd` array. This variable is read-only, and it may change after calls to `svc_getreg_poll()` or any creation routines.

`bool_t svc_freeargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using `svc_getargs()` . This routine returns `TRUE` if the results were successfully freed, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`bool_t svc_getargs(const SVCXPRT * xprt , const xdrproc_t inproc , caddr_t in);`
 A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt* . The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns `TRUE` if decoding succeeds, and `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User` MT modes.

`void svc_getreq_common(const int fd);`
 This routine is called to handle a request on the given file descriptor.

`void svc_getreq_poll(struct pollfd * pfds , const int pollretval);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `poll(2)` has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from `poll(2)` and *pfds* is the array of `pollfd` structures on which the `poll(2)` was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

`void svc_getreqset(fd_set * rdfds);`
 This routine is only of interest if a service implementor does not call `svc_run()` , but instead implements custom asynchronous event processing. It is called when `select(3C)` has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

`struct netbuf *svc_getrpcaller(const SVCXPRT * xprt);`
 The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt* .

This function macro is safe in MT applications.

void svc_run(void);

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using `svc_getreq_poll()` when one arrives. This procedure is usually waiting for the `poll(2)` library call to return.

Applications executing in the `Automatic` or `User MT` modes should invoke this function exactly once. In the `Automatic MT` mode, it will create threads to service client requests. In the `User MT` mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

bool_t svc_sendreply(const SVCXPRT * *xprt*, const xdrproc_t *outproc*, const caddr_t *out*);

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns `TRUE` if it succeeds, `FALSE` otherwise.

This function macro is safe in MT applications utilizing the `Automatic` or `User MT` modes.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	See NOTES below.

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when a `bind(3N)` call is made, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL`. If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control

structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`rpc(3N)`, `rpc_svc_create(3N)`, `rpc_svc_reg(3N)`, `libt6(3N)`,
`t6alloc_blk(3N)`, `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

`rpcgen(1)`, `poll(2)`, `rpc_control(3N)`, `rpc_svc_err(3N)`, `select(3C)`,
`xprt_register(3N)`, `attributes(5)`

NOTES

`svc_dg_enablecache()` and `svc_getrpccaller()` are safe in multithreaded applications. `svc_freeargs()`, `svc_getargs()`, and `svc_sendreply()` are safe in MT applications utilizing the Automatic or User MT modes. `svc_getreq_common()`, `svc_getreqset()`, and `svc_getreq_poll()` are unsafe in multithreaded applications and should be called only from the main thread.

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT * svc, const uint_t req, void * info);</code></p> <p>A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*.

int *svc_create*(const void (* *dispatch*)(const struct *svc_req* *, const SVCXPRT *), const *rpcprog_t* *prognum*, const *rpcvers_t* *versnum*, const char * *nettype*);
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in *NETPATH* variable or in top to bottom order in the *netconfig* database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

void *svc_destroy*(SVCXPRT * *xprt*);

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

SVCXPRT **svc_dg_create*(const *int* *fildez*, const *uint_t* *sendsz*, const *uint_t* *recvsz*);

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildes* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildes*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns *NULL* if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see *clnt_raw_create()* in *rpc_clnt_create(3N)*). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns *NULL* if it fails, and an error message is logged.

Note: *svc_run()* should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildes*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildes* is the file descriptor on which the service is listening. If *fildes* is *RPC_ANYFD*, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildes* is bound to the address specified by *bindaddr*, otherwise *fildes* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns *NULL* if it fails, and an error message is logged. The server is not registered with the *rpcbind(1M)* service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

svc_tp_create() creates a server handle for the network specified by *netconf*, and registers itself with the *rpcbind* service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev, const uint_t sendsz, const uint_t recvsz);`
 This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N) , attributes(5)

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT *svc, const uint_t req, void *info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*

int svc_create(const void (dispatch)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);*
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

*void svc_destroy(SVCXPRT * *xprt*);*

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

*SVCXPRT *svc_dg_create(const int *fildez*, const uint_t *sendsz*, const uint_t *recvsz*);*

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *rcvsvz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildev* should be open and bound. The server is not registered with *rpcbind(1M)*.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildev*, const uint_t *sendsz*, const uint_t *rcvsvz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *rcvsvz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns *NULL* if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see *clnt_raw_create()* in *rpc_clnt_create(3N)*). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns *NULL* if it fails, and an error message is logged.

Note: *svc_run()* should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildev*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *rcvsvz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildev* is the file descriptor on which the service is listening. If *fildev* is *RPC_ANYFD*, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildev* is bound to the address specified by *bindaddr*, otherwise *fildev* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *rcvsvz*; values of 0 choose suitable defaults. This routine returns *NULL* if it fails, and an error message is logged. The server is not registered with the *rpcbind(1M)* service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

svc_tp_create() creates a server handle for the network specified by *netconf*, and registers itself with the *rpcbind* service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev , const uint_t sendsz , const uint_t recvsz);`
This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO

Trusted Solaris 7
Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N), attributes(5)

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
DESCRIPTION	<p>These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code>], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (* procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char * nettype);</code></p> <p>Register program <i>prognum</i>, procedure <i>procname</i>, and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i>, version <i>versnum</i>, and procedure <i>procnum</i>, <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its <code>static</code> result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters while <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i>. See <code>rpc(3N)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p> <p><code>int svc_reg(const SVCXPRT * xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (* dispatch)(), const struct netconfig * netconf);</code></p> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i> \Rightarrow <i>nc_netid</i>] to <i>xpirt</i> \Rightarrow <i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p>

```
void svc_unreg(const rpcprog_t prognum , const rpcvers_t versnum );
```

Remove from the `rpcbind` service, all mappings of the triple [*prognum* , *versnum* , *all-transport*] to network address and all mappings within the RPC service package of the double [*prognum* , *versnum*] to dispatch routines.

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

The `PRIV_NET_SETID` privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

```
int svc_auth_reg(const int cred_flavor , const enum auth_stat (*handler)());
```

Registers the service authentication routine *handler* with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type *cred_flavor* . This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call `svc_auth_reg()` after registering the service and prior to calling `svc_run()` . When needed to process an RPC credential of type *cred_flavor* , the *handler* procedure will be called with two parameters (`struct svc_req * rqst` , `struct rpc_msg * msg`) and is expected to return a valid `enum auth_stat` value. There is no provision to change or delete an authentication handler once registered.

The `svc_auth_reg()` routine returns 0 if the registration is successful, 1 if *cred_flavor* already has an authentication handler registered for it, and -1 otherwise.

```
void xpirt_register(const SVCXPRT * xprt );
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3N)`). Service implementors usually do not need this routine.

```
void xpirt_unregister(const SVCXPRT * xprt );
```

Before an RPC service transport handle *xprt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable `svc_fdset` [see `rpc_svc_calls(3N)`]. Service implementors usually do not need this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is on when `rpc_reg()` or `rpc_svc()` is called, a multilevel mapping is created. To delete a multilevel mapping, `svc_unreg()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The `PRIV_NET_SETID` privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`inetd(1M)`, `rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`,
`rpc_svc_create(3N)`, `rpcbind(3N)`

`select(3C)`, `rpc_svc_err(3N)`, `attributes(5)`

NAME	rpc_svc_create, svc_control, svc_create, svc_destroy, svc_dg_create, svc_fd_create, svc_raw_create, svc_tli_create, svc_tp_create, svc_vc_create – Library routines for the creation of server handles				
DESCRIPTION	<p>These routines are part of the RPC library which allows C language programs to make procedure calls on servers across the network. These routines deal with the creation of service handles. Once the handle is created, the server can be invoked by calling <code>svc_run()</code>.</p> <p>Privileged programs can create multilevel ports, create multilevel mappings, and access network security attributes. See SUMMARY OF TRUSTED SOLARIS CHANGES for more information.</p>				
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t svc_control(SVCXPRT *svc, const uint_t req, void *info);</code> A function to change or retrieve various information about a service object. <i>req</i> indicates the type of operation and <i>info</i> is a pointer to the information. The supported values of <i>req</i>, their argument types, and what they do are:</p> <table> <tr> <td><code>SVCGET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.</td></tr> <tr> <td><code>SVCSET_VERSQUIET</code></td><td>If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.</td></tr> </table>	<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.	<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.
<code>SVCGET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. <i>info</i> should be a pointer to an integer. Upon successful completion of the <code>SVCGET_VERSQUIET</code> request, * <i>info</i> contains an integer which describes the server's current behavior: 0 indicates normal server behavior, that is, an <code>RPC_PROGVERSMISMATCH</code> error will be returned; 1 indicates that the out of range request will be silently ignored.				
<code>SVCSET_VERSQUIET</code>	If a request is received for a program number served by this server but the version number is outside the range registered with the server, an <code>RPC_PROGVERSMISMATCH</code> error will normally be returned. It is sometimes desirable to change this behavior. <i>info</i> should be a pointer to an integer which is either 0, indicating normal server behavior and an <code>RPC_PROGVERSMISMATCH</code> error will be returned, or -1, indicating that the out of range request should be silently ignored.				

SVCGET_XID

Returns the transaction ID of connection-oriented (vc) and connectionless (dg) transport service calls. The transaction ID assists in uniquely identifying client requests for a given RPC version, program number, procedure, and client. The transaction ID is extracted from the service transport handle *svc*; *info* must be a pointer to an unsigned long. Upon successful completion of the SVCGET_XID request, * *info* contains the transaction ID. Note that rendezvous and raw service handles do not define a transaction ID. Thus, if the service handle is of rendezvous or raw type, and the request is of type SVCGET_XID, *svc_control()* will return FALSE. Note also that the transaction ID read by the server can be set by the client through the suboption CLSET_XID in *clnt_control()*. See *clnt_create(3N)*

int *svc_create*(const void (* *dispatch*)(const struct *svc_req* *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const char * *nettype*);
svc_create() creates server handles for all the transports belonging to the class *nettype*.

nettype defines a class of transports which can be used for a particular application. The transports are tried in left to right order in NETPATH variable or in top to bottom order in the netconfig database. If *nettype* is NULL, it defaults to *netpath*.

svc_create() registers itself with the *rpcbind* service [see *rpcbind(1M)*]. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*; this requires calling *svc_run()* (see *svc_run()* in *rpc_svc_reg(3N)*). If *svc_create()* succeeds, it returns the number of server handles it created, otherwise it returns 0 and an error message is logged.

void *svc_destroy*(SVCXPRT * *xprt*);

A function macro that destroys the RPC service handle *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

SVCXPRT **svc_dg_create*(const int *fildez*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a connectionless RPC service handle, and returns a pointer to it. This routine returns NULL if it fails, and an error message is

logged. *sendsz* and *recvsz* are parameters used to specify the size of the buffers. If they are 0, suitable defaults are chosen. The file descriptor *fildev* should be open and bound. The server is not registered with `rpcbind(1M)`.

Warning: since connectionless-based RPC messages can only hold limited amount of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

SVCXPRT *svc_fd_create(const int *fildev*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates a service on top of an open and bound file descriptor, and returns the handle to it. Typically, this descriptor is a connected file descriptor for a connection-oriented transport. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are 0, reasonable defaults are chosen. This routine returns `NULL` if it fails, and an error message is logged.

SVCXPRT *svc_raw_create(void);

This routine creates an RPC service handle and returns a pointer to it. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; (see `clnt_raw_create()` in `rpc_clnt_create(3N)`). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel and networking interference. This routine returns `NULL` if it fails, and an error message is logged.

Note: `svc_run()` should not be called when the raw interface is being used.

SVCXPRT *svc_tli_create(const int *fildev*, const struct netconfig * *netconf*, const struct t_bind * *bindaddr*, const uint_t *sendsz*, const uint_t *recvsz*);

This routine creates an RPC server handle, and returns a pointer to it. *fildev* is the file descriptor on which the service is listening. If *fildev* is `RPC_ANYFD`, it opens a file descriptor on the transport specified by *netconf*. If the file descriptor is unbound and *bindaddr* is non-null *fildev* is bound to the address specified by *bindaddr*, otherwise *fildev* is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of 0 choose suitable defaults. This routine returns `NULL` if it fails, and an error message is logged. The server is not registered with the `rpcbind(1M)` service.

SVCXPRT *svc_tp_create(const void (* *dispatch*)(const struct svc_req *, const SVCXPRT *), const rpcprog_t *prognum*, const rpcvers_t *versnum*, const struct netconfig * *netconf*);

`svc_tp_create()` creates a server handle for the network specified by *netconf*, and registers itself with the `rpcbind` service. *dispatch* is called

when there is a remote procedure call for the given *prognum* and *versnum* ; this requires calling `svc_run()` . `svc_tp_create()` returns the service handle if it succeeds, otherwise a `NULL` is returned and an error message is logged.

`SVCXPRT *svc_vc_create(const int fildev , const uint_t sendsz , const uint_t rcvsvsz);`
 This routine creates a connection-oriented RPC service and returns a pointer to it. This routine returns `NULL` if it fails, and an error message is logged. The users may specify the size of the send and receive buffers with the parameters *sendsz* and *rcvsvsz* ; values of 0 choose suitable defaults. The file descriptor *fildev* should be open and bound. The server is not registered with the `rpcbind(1M)` service.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The `PRIV_NET_MAC_READ` privilege affects the operation of trusted network services for binding to transport addresses. If the privilege is on when an RPC library routine such as `svc_create()` binds to a transport, a multilevel port will be created.

Most `rpcbind()` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind()` services. If the privilege is on when a library routine calls `rpcbind()` to create a mapping, a multilevel mapping is created.

The `PRIV_NET_PRIVADDR` privilege is required when a library routine calls `rpcbind()` to create a mapping for a transport that uses a privileged address.

The `SVCXPRT` structure allows a server to provide `t6attr_t` pointers to opaque structures for receiving security attributes with a client request or setting the security attributes of a reply. When a new `SVCXPRT` structure is created, the pointers are initialized to `NULL` . If it needs to access the security attributes, the server must use the `t6alloc_blk()` routine to allocate attribute-control structures and set the `t6attr_t` pointers in the `SVCXPRT` structure. When `svc_destroy()` is used to destroy a service handle, the server should also use `t6free_blk()` to free any attribute-control structures previously allocated for that service handle.

SEE ALSO Trusted Solaris 7 Reference Manual

`rpcbind(1M)` , `rpc(3N)` , `rpc_clnt_create(3N)` , `rpc_svc_calls(3N)` ,
`rpc_svc_reg(3N)` , `libt6(3N)` , `t6alloc_blk(3N)` , `t6free_blk(3N)`

**SunOS 5.7 Reference
Manual**

rpc_svc_err(3N) , attributes(5)

NAME	t6alloc_blk, t6free_blk – Allocate and free security-attribute control structure and buffer						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> t6attr_t t6alloc_blk(t6mask_t mask); void t6free_blk(t6attr_t t6ctl);</pre>						
DESCRIPTION	<p>t6alloc_blk() allocates a t6attr_t structure, which is an opaque handle used to access the full set of security attributes supported on the system. See man pages for t6get_attr(3N) and t6set_attr(3N) for more information on how generic TSIX application programs can access security attributes referenced by t6attr_t without knowing how references are built between the t6attr_t structure and the sets of security attributes for each individual TSIX operating system vendor. If t6alloc_blk() is successful, the opaque handle is returned that can be used to set or get individual attributes to or from this control structure. t6alloc_blk() allocates space in the control structure for all attributes specified in the <i>mask</i> parameter.</p> <p>t6free_blk() should be used in conjunction with t6alloc_blk() to free the opaque control structure and any space within it.</p>						
RETURN VALUES	Upon successful completion, t6alloc_blk() returns a pointer to the t6attr_t structure. Upon failure, t6alloc_blk() returns a NULL pointer.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N) , t6get_attr(3N) , t6set_attr(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
WARNINGS	For generic TSIX applications, use t6free_blk() to free memory allocated by t6alloc_blk() for better portability.						
NOTES	This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; this interface is available in TSIX (RE) 1.1- API -compliant systems.						

NAME	t6attr_query – Get mask indicating which attributes came from templates
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6attr_query(int fd, t6mask_t *mask);</pre>
DESCRIPTION	<p>Not all security attributes are transmitted with the data. Missing security attributes are taken from a database on the receiving machine. t6attr_query() allows a process to determine when a security attribute comes from a database. A bit value of 1 in the mask indicates that the attribute comes from a database.</p>
RETURN VALUES	<p>t6attr_query() returns:</p> <ul style="list-style-type: none">0 On success.-1 If an error is encountered.
NOTES	<p>This interface is specific to the Trusted Solaris environment.</p>

NAME	t6clear_blk – Clear security attributes						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> void t6clear_blk(t6mask_t mask, t6attr_t src);</pre>						
DESCRIPTION	t6clear_blk() clears attributes specified in <i>mask</i> from the t6attr_t control structure <i>src</i> , which is passed in as an argument.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO Trusted Solaris 7 Reference Manual SunOS 5.7 Reference Manual	libt6(3N), t6alloc_blk(3N) attributes(5)						
NOTES	<p>This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; this interface is available in TSIX(RE) 1.1-API-compliant systems.</p>						

NAME	t6cmp_blk – Compare security attributes						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6cmp_blk(const t6attr_t <i>ctl1</i>, const t6attr_t <i>ctl2</i>);</pre>						
DESCRIPTION	t6cmp_blk() compares two t6attr_t control structures for the attributes contained. The two t6attr_t control structures are regarded as equal if each type of attribute that exists in <i>ctl1</i> exists in <i>ctl2</i> and vice versa, and if the attribute values of each type in <i>ctl1</i> and <i>ctl2</i> are the same.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	This call returns zero if <i>ctl1</i> equals <i>ctl2</i> ; otherwise, the call returns a nonzero value.						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N), t6alloc_blk(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
NOTES	This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.						

NAME t6copy_blk – Copy security attributes

SYNOPSIS `cc [flags...] file ... -lt6`

```
#include <tsix/t6attrs.h>
void t6copy_blk(const t6attr_t attr1, t6attr_t attr2);
```

DESCRIPTION t6copy_blk() copies a set of attributes specified by *attr1* into the buffers controlled by *attr2* after both *attr1* and *attr2* have been allocated by t6alloc_blk(). See man pages for t6alloc_blk(3N) for more details.

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

SEE ALSO

Trusted Solaris 7
Reference Manual

libt6(3N), t6alloc_blk(3N)

SunOS 5.7 Reference
Manual

attributes(5)

NOTES

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

NAME	t6dup_blk – Duplicate security attributes						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> t6attr_t t6dup_blk(const t6attr_t src);</pre>						
DESCRIPTION	<p>t6dup_blk() allocates a new t6attr_t control structure and buffer space large enough to hold the set of security attributes in the t6attr_t control structure src, which is passed in as an argument. t6dup_blk() then copies that set of attributes specified by src into the newly allocated structure. Upon successful completion, the newly created t6attr_t handle is returned.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>Upon successful completion, t6attr_t returns a new handle. If it is unable to allocate sufficient memory for the new attributes, t6dup() returns NULL and sets errno to an appropriate value.</p>						
DIAGNOSTICS	<p>ENOMEM Out of memory for allocation</p>						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N), t6alloc_blk(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
NOTES	<p>This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.</p>						

NAME	t6ext_attr, t6new_attr – Manipulate network-endpoint security options						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6ext_attr(int fd, t6cmd_t cmd); int t6new_attr(int fd, t6cmd_t cmd);</pre>						
DESCRIPTION	<p>t6ext_attr() turns on extended security operations on the trusted IPC mechanism. <i>fd</i> is the descriptor associated with the IPC mechanism. <i>cmd</i> must be either ON to turn on extended operations or OFF to turn them off. When first created, the trusted IPC mechanism appears the same as an untrusted IPC mechanism. The trusted mechanism can be used in the same way to send and receive data as long as communications do not violate the security policies of the system. Between systems that support mandatory access control, for example, communications can occur only between processes at the same sensitivity level. Before it allows a process to specify security attributes or manipulate the endpoint's security options, the network endpoint must call t6ext_attr() .</p> <p>t6new_attr() with a <i>cmd</i> value of ON tells the underlying TSIX software that the receiving process is interested in security attributes only if they differ from the last set of attributes received. After this call, t6recvfrom(3N) returns valid security attributes only when a change in the attributes is detected. This situation is indicated by setting the t6recvfrom() parameter <i>new_attrs</i> to nonzero. When new attributes are returned, the full set of requested attributes is returned, not just those that have changed. When <i>cmd</i> is OFF , the default situation prevails: attributes are returned with each call to t6recvfrom() .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3) , t6recvfrom(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
NOTES	<p>In the Trusted Solaris environment, t6ext_attr() is a NULL function.</p> <p>This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1- API -compliant systems.</p>						

NAME	t6alloc_blk, t6free_blk – Allocate and free security-attribute control structure and buffer						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> t6attr_t t6alloc_blk(t6mask_t mask); void t6free_blk(t6attr_t t6ctl);</pre>						
DESCRIPTION	<p>t6alloc_blk() allocates a t6attr_t structure, which is an opaque handle used to access the full set of security attributes supported on the system. See man pages for t6get_attr(3N) and t6set_attr(3N) for more information on how generic TSIX application programs can access security attributes referenced by t6attr_t without knowing how references are built between the t6attr_t structure and the sets of security attributes for each individual TSIX operating system vendor. If t6alloc_blk() is successful, the opaque handle is returned that can be used to set or get individual attributes to or from this control structure. t6alloc_blk() allocates space in the control structure for all attributes specified in the <i>mask</i> parameter.</p> <p>t6free_blk() should be used in conjunction with t6alloc_blk() to free the opaque control structure and any space within it.</p>						
RETURN VALUES	Upon successful completion, t6alloc_blk() returns a pointer to the t6attr_t structure. Upon failure, t6alloc_blk() returns a NULL pointer.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N) , t6get_attr(3N) , t6set_attr(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
WARNINGS	For generic TSIX applications, use t6free_blk() to free memory allocated by t6alloc_blk() for better portability.						
NOTES	This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; this interface is available in TSIX (RE) 1.1- API -compliant systems.						

NAME	t6get_attr, t6set_attr – Get security attributes from or set security attributes in the security-attribute buffer handled by a control structure						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> void * t6get_attr(t6attr_id_t attr_type, const t6ctrl_t t6ctl, t6attr_t t6ctl); int t6set_attr(t6attr_id_t attr_type, const void * attr_buf);</pre>						
DESCRIPTION	<p>t6get_attr() takes a control structure, <i>t6ctl</i>, and attribute type, <i>attr_type</i>, and returns a pointer to the requested attribute value (type) from the opaque control structure <i>t6ctl</i>. <i>attr_type</i> contains a number (defined in <tsix/t6attrs.h>) that specifies which type of attribute the caller is interested in getting. Only one type can be specified per call.</p> <p>Returned value by t6get_attr() should be type cast to the standard type that represents the type indicated by <i>attr_type</i>.</p> <p>t6set_attr() replaces the requested attribute value (type) in <i>t6ctl</i> with the value to which <i>attr_buf</i> points. The type of the attribute is specified in <i>attr_type</i> as one of the numbers defined in <tsix/t6attrs.h>.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>Upon successful completion, t6get_attr() returns a pointer to the appropriate value if it exists in the attribute structure. Upon failure, t6get_attr() returns NULL. t6set_attr() returns 0 if the attribute structure can contain the requested attribute; if not, t6set_attr() returns -1 and does not change the attribute structure.</p>						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N), t6alloc_blk(3N), t6free_blk(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
NOTES	<p>In the Trusted Solaris environment, t6get_attr() returns values of these types:</p> <p>au_id_t Audit ID</p>						

<code>auditinfo_t</code>	Audit info
<code>bclear_t</code>	Clearance
<code>bslabel_t</code>	Sensitivity label
<code>gid_t</code>	Effective group ID
<code>gid_t</code>	Supplemental group ID s
<code>pattr_t</code>	Process attributes
<code>priv_set_t</code>	Effective privileges
<code>sid_t</code>	Session ID
<code>pid_t</code>	Process ID
<code>uid_t</code>	Effective user ID

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1- API -compliant systems.

NAME	t6get_endpt_mask, t6set_endpt_mask, t6get_endpt_default, t6set_endpt_default – Get and set endpoint mask, or get and set endpoint default attributes
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6get_endpt_mask(int fd, t6mask_t * mask); int t6set_endpt_mask(int fd, t6mask_t mask); int t6get_endpt_default(int fd, t6mask_t * mask, t6attr_t attr_ptr); int t6set_endpt_default(int fd, t6mask_t mask, const t6attr_t attr_ptr);</pre>
DESCRIPTION	<p>The security extensions on the communication endpoint include a set of default security attributes that may be applied to outgoing data and an attribute mask that designates which attributes are taken from the endpoint's default attributes and which are taken from the process' effective attributes.</p> <p>By default, data written to an endpoint has associated with it the security attributes of the process that wrote the data. However, a privileged process may change the value of the default attribute mask on an endpoint the process had created, and the endpoint's default attributes.</p> <p>t6get_endpt_mask() allows a process to obtain the current setting of the default attribute mask for the endpoint specified by <i>fd</i> . The attribute mask is returned in the parameter <i>mask</i> .</p> <p>t6set_endpt_mask() allows a process to set the bit values of the default attribute mask for the endpoint specified by <i>fd</i> to the value specified by <i>mask</i> . A bit value of 0 indicates the attribute is taken from the process's effective attributes; and a bit value of 1 indicates the attribute is taken from the endpoint's default attributes.</p> <p>t6get_endpt_default() allows a process to get the current setting of the default attributes of the endpoint specified by <i>fd</i> . <i>mask</i> indicates which attributes are present in the <i>attr_ptr</i> parameter. To access <i>attr_ptr</i> , see t6get_attr(3N) .</p> <p>t6set_endpt_default() allows a process to set the default attributes of the endpoint specified by <i>fd</i> to the attributes specified by <i>attr_ptr</i> . <i>mask</i> indicates which attributes are present in <i>attr_ptr</i> . To set up <i>attr_ptr</i> , see t6set_attr(3N) .</p> <p>Only a process with the appropriate override privileges can change the endpoint's attribute mask or default attributes. To change an endpoint's default attribute or its mask bit, a process must have the override privilege corresponding to the attribute. The override privilege required to specify a default attribute is implementation-specific.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

Upon successful completion, these calls return 0 . If either call encounters an error, the call returns -1 .

SEE ALSO

**Trusted Solaris 7
Reference Manual**

libt6(3N) , t6sendto(3N) , t6set_attr(3N)

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX (RE) 1.1- API -compliant systems.

NAME	t6get_endpt_mask, t6set_endpt_mask, t6get_endpt_default, t6set_endpt_default – Get and set endpoint mask, or get and set endpoint default attributes
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6get_endpt_mask(int fd, t6mask_t * mask); int t6set_endpt_mask(int fd, t6mask_t mask); int t6get_endpt_default(int fd, t6mask_t * mask, t6attr_t attr_ptr); int t6set_endpt_default(int fd, t6mask_t mask, const t6attr_t attr_ptr);</pre>
DESCRIPTION	<p>The security extensions on the communication endpoint include a set of default security attributes that may be applied to outgoing data and an attribute mask that designates which attributes are taken from the endpoint's default attributes and which are taken from the process' effective attributes.</p> <p>By default, data written to an endpoint has associated with it the security attributes of the process that wrote the data. However, a privileged process may change the value of the default attribute mask on an endpoint the process had created, and the endpoint's default attributes.</p> <p>t6get_endpt_mask() allows a process to obtain the current setting of the default attribute mask for the endpoint specified by <i>fd</i> . The attribute mask is returned in the parameter <i>mask</i> .</p> <p>t6set_endpt_mask() allows a process to set the bit values of the default attribute mask for the endpoint specified by <i>fd</i> to the value specified by <i>mask</i> . A bit value of 0 indicates the attribute is taken from the process's effective attributes; and a bit value of 1 indicates the attribute is taken from the endpoint's default attributes.</p> <p>t6get_endpt_default() allows a process to get the current setting of the default attributes of the endpoint specified by <i>fd</i> . <i>mask</i> indicates which attributes are present in the <i>attr_ptr</i> parameter. To access <i>attr_ptr</i> , see t6get_attr(3N) .</p> <p>t6set_endpt_default() allows a process to set the default attributes of the endpoint specified by <i>fd</i> to the attributes specified by <i>attr_ptr</i> . <i>mask</i> indicates which attributes are present in <i>attr_ptr</i> . To set up <i>attr_ptr</i> , see t6set_attr(3N) .</p> <p>Only a process with the appropriate override privileges can change the endpoint's attribute mask or default attributes. To change an endpoint's default attribute or its mask bit, a process must have the override privilege corresponding to the attribute. The override privilege required to specify a default attribute is implementation-specific.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

Upon successful completion, these calls return 0 . If either call encounters an error, the call returns -1 .

SEE ALSO

**Trusted Solaris 7
Reference Manual**

libt6(3N) , t6sendto(3N) , t6set_attr(3N)

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX (RE) 1.1- API -compliant systems.

NAME	t6peek_attr, t6last_attr – Examine the security attributes on the next or the previous byte of data						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6peek_attr(int fd, t6attr_t attr_ptr, t6mask_t * new_attrs); int t6last_attr (int fd, t6attr_t attr_ptr, t6mask_t * new_attrs);</pre>						
DESCRIPTION	<p>t6peek_attr() allows a process to peek ahead at the security attributes of the next byte of data. <i>fd</i> is the descriptor of the endpoint; <i>attr_ptr</i> specifies a structure in which to store those attributes the caller wishes to retrieve. <i>new_attrs</i> points to a mask that indicates which attributes were actually retrieved on return from t6peek_attr() .</p> <p>t6last_attr() allows a process to retrieve the attributes of the last byte of data read from the indicated file descriptor. The parameters for t6last_attr() are identical to those for the t6peek_attr() routine.</p> <p>If no messages are available at the socket, the examining call waits for a message to arrive, unless the socket is non-blocking (see <i>fcntl(2)</i>), in which case -1 is returned with the external variable <i>errno</i> set to <i>EWOULDBLOCK</i> .</p>						
RETURN VALUES	Upon successful completion, these calls return 0 , place the retrieved security attributes in the <i>t6attr_t</i> structure, and set <i>*new_attrs</i> to the mask of those attributes actually returned. If either call encounters an error, it returns -1 . When they generate errors, t6peek_attr() and t6last_attr() set <i>errno</i> .						
ATTRIBUTES	See <i>attributes(5)</i> for descriptions of the following attributes:						
	<table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N) , <i>fcntl(2)</i>						
SunOS 5.7 Reference Manual	<i>attributes(5)</i>						
NOTES	This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1- API -compliant systems.						

NAME	t6ext_attr, t6new_attr – Manipulate network-endpoint security options						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6ext_attr(int fd, t6cmd_t cmd); int t6new_attr(int fd, t6cmd_t cmd);</pre>						
DESCRIPTION	<p>t6ext_attr() turns on extended security operations on the trusted IPC mechanism. <i>fd</i> is the descriptor associated with the IPC mechanism. <i>cmd</i> must be either ON to turn on extended operations or OFF to turn them off. When first created, the trusted IPC mechanism appears the same as an untrusted IPC mechanism. The trusted mechanism can be used in the same way to send and receive data as long as communications do not violate the security policies of the system. Between systems that support mandatory access control, for example, communications can occur only between processes at the same sensitivity level. Before it allows a process to specify security attributes or manipulate the endpoint's security options, the network endpoint must call t6ext_attr() .</p> <p>t6new_attr() with a <i>cmd</i> value of ON tells the underlying TSIX software that the receiving process is interested in security attributes only if they differ from the last set of attributes received. After this call, t6recvfrom(3N) returns valid security attributes only when a change in the attributes is detected. This situation is indicated by setting the t6recvfrom() parameter <i>new_attrs</i> to nonzero. When new attributes are returned, the full set of requested attributes is returned, not just those that have changed. When <i>cmd</i> is OFF , the default situation prevails: attributes are returned with each call to t6recvfrom() .</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3) , t6recvfrom(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
NOTES	<p>In the Trusted Solaris environment, t6ext_attr() is a NULL function.</p> <p>This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1- API -compliant systems.</p>						

NAME	t6peek_attr, t6last_attr – Examine the security attributes on the next or the previous byte of data						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6peek_attr(int fd, t6attr_t attr_ptr, t6mask_t * new_attrs); int t6last_attr (int fd, t6attr_t attr_ptr, t6mask_t * new_attrs);</pre>						
DESCRIPTION	<p>t6peek_attr() allows a process to peek ahead at the security attributes of the next byte of data. <i>fd</i> is the descriptor of the endpoint; <i>attr_ptr</i> specifies a structure in which to store those attributes the caller wishes to retrieve. <i>new_attrs</i> points to a mask that indicates which attributes were actually retrieved on return from t6peek_attr() .</p> <p>t6last_attr() allows a process to retrieve the attributes of the last byte of data read from the indicated file descriptor. The parameters for t6last_attr() are identical to those for the t6peek_attr() routine.</p> <p>If no messages are available at the socket, the examining call waits for a message to arrive, unless the socket is non-blocking (see <i>fcntl(2)</i>), in which case -1 is returned with the external variable <i>errno</i> set to <i>EWOULDBLOCK</i> .</p>						
RETURN VALUES	Upon successful completion, these calls return 0 , place the retrieved security attributes in the t6attr_t structure, and set *new_attrs to the mask of those attributes actually returned. If either call encounters an error, it returns -1 . When they generate errors, t6peek_attr() and t6last_attr() set <i>errno</i> .						
ATTRIBUTES	See <i>attributes(5)</i> for descriptions of the following attributes:						
	<table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N) , <i>fcntl(2)</i>						
SunOS 5.7 Reference Manual	<i>attributes(5)</i>						
NOTES	This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1- API -compliant systems.						

NAME t6recvfrom – Read security attributes and data from a trusted endpoint

SYNOPSIS `cc [flags...] file ... -lsocket -lnsl -lt6 [library...]`

```
#include <tsix/t6attrs.h>
int t6recvfrom(int fd, char *buf, int len, int flags, struct sockaddr *from, int *fromlen,
t6attr_t attr_ptr, t6mask_t *new_attrs);
```

DESCRIPTION

t6recvfrom() receives data and its associated security attributes from a communication endpoint. The *from* and *fromlen* parameters are used only if you wish to receive the source address for the data. This information may not be applicable for some trusted endpoints. If not used, these fields should be set to 0. If *from* is not a NULL pointer, the source address of the message is filled in. *fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket from which the message is received. [See socket(3N).]

The *flags* parameter is formed by ORing one or more of these values:

MSG_OOB Read any out-of-band data present on the socket rather than the regular in-band data. If *attr_ptr* is not NULL, out-of-band data security attributes are also retrieved.

MSG_PEEK Peek at the data present on the socket; the data is returned but not consumed, so that a subsequent receive operation will see the same data. If *attr_ptr* is not NULL, security attributes of the data are also peeked.

attr_ptr specifies a control structure in which to store those attributes the caller wishes to retrieve. To get an attribute from the control structure, see t6get_attr(3N). Any attribute that the receiving process does not care to receive may not be specified in the control structure. This selectivity minimizes the attribute-translation time when passing the attributes out of the kernel.

If the t6new_attr(3N) call was made previously with a setting of ON, the security attributes of the received data will be returned only if they have changed from the last set read. **new_attrs* is set to the mask of those attributes actually returned. If new attributes are detected, all attributes requested by the receiving process are returned, not just those that have changed.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

Upon success, `t6recvfrom()` returns the number of bytes read. Upon failure, `t6recvfrom()` returns `-1` and sets `errno`.

Always checking the return value is critical. Revocation of access is possible if the received data changes to a level not accessible to the receiving process.

ERRORS

The calls fail if any of these conditions is true:

<code>EBADF</code>	<code>fd</code> is an invalid file descriptor.
<code>EINTR</code>	The operation was interrupted by delivery of a signal before any data was available to be received.
<code>EIO</code>	An I/O error occurred while reading from or writing to the file system.
<code>ENOMEM</code>	There was insufficient user memory available for the operation to complete.
<code>ENOSR</code>	There were insufficient STREAMS resources available for the operation to complete.
<code>ENOTSOCK</code>	<code>fd</code> is not a socket.
<code>ESTALE</code>	A stale NFS file handle exists.

SEE ALSO

`libt6(3N)`, `t6get_attr(3N)`, `t6sendto(3N)`

NOTES

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

Only `SOCK_STREAM` sockets created in the `AF_INET` address family support out-of-band data.

NAME	t6sendto – Specify security attributes to send with data on a trusted endpoint				
SYNOPSIS	<pre>cc [flags...] file ... -lsocket -lnsl -lt6 [library...]</pre>				
DESCRIPTION	<pre>#include <tsix/t6attrs.h> int t6sendto(int fd, const char *msg, int len, int flags, const struct sockaddr *to, int tolen, const t6attr_t attr_ptr);</pre> <p>t6sendto() allows a privileged process to specify the security attributes to send with an IPC message. A process may specify only those attributes for which it possesses the appropriate override privilege and need not specify a full set. Any unspecified attributes are supplied by the kernel.</p> <p><i>fd</i> is a socket created with socket(3N). The address of the target is given by <i>to</i> with <i>tolen</i> specifying its size. The length of the message is given by <i>len</i>.</p> <p>The <i>to</i> pointer and <i>to_len</i> parameter are used only if you are specifying the destination address; otherwise they should be set to 0. You may not specify the address if the trusted endpoint was created for a connection-oriented protocol, such as TCP. If the message is too long to pass atomically through the underlying protocol, then the message is not transmitted and the error EMSGSIZE is returned.</p> <p>A return value of -1 indicates locally detected errors only, not implicitly that the message was not delivered.</p> <p>The <i>flags</i> parameter is formed from the bitwise OR of zero or more of these values:</p> <table> <tr> <td>MSG_OOB</td><td>Send out-of-band data and any security attributes specified by a privileged process on sockets that support this notion provided that the underlying protocol also supports out-of-band data. Data and attributes sent with this flag are typically not subject to the internal buffering normally applied by the network to improve network efficiency.</td></tr> <tr> <td>MSG_DONTROUTE</td><td>The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.</td></tr> </table> <p>The security attributes are specified by the <i>attr_ptr</i> parameter. To set up <i>attr_ptr</i>, see t6set_attr(3N).</p> <p>Only a process with the appropriate override privileges can specify the security attributes associated with the data it sends. To specify an attribute, a process must have the override privilege corresponding to the attribute. The override privilege required to specify an attribute is implementation specific. For Trusted Solaris, one or more of these privileges may be required: PRIV_NET_DOWNGRADE_SL, PRIV_NET_UPGRADE_SL,</p>	MSG_OOB	Send out-of-band data and any security attributes specified by a privileged process on sockets that support this notion provided that the underlying protocol also supports out-of-band data. Data and attributes sent with this flag are typically not subject to the internal buffering normally applied by the network to improve network efficiency.	MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.
MSG_OOB	Send out-of-band data and any security attributes specified by a privileged process on sockets that support this notion provided that the underlying protocol also supports out-of-band data. Data and attributes sent with this flag are typically not subject to the internal buffering normally applied by the network to improve network efficiency.				
MSG_DONTROUTE	The SO_DONTROUTE option is turned on for the duration of the operation. This option is used only by diagnostic or routing programs.				

PRIV_NET_DOWNGRADE_IL, PRIV_NET_UPGRADE_IL, PRIV_NET_SETCLR, PRIV_NET_SETID, PRIV_NET_SETPRIV, PRIV_NET_BROADCAST. Information Labels (ILs) are now obsolete. See NOTES.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

Upon success, the return value is the number of bytes actually sent. Upon failure, the call returns `-1` and sets the error code in `errno`.

Always checking the return value is critical, for the addition of security means that access to an endpoint may be revoked in response to a security violation.

ERRORS

`t6sendto()` fails if any of these conditions is true:

EBADF	<i>fd</i> is an invalid file descriptor.
EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL	<i>to</i> len is not the size of a valid address for the specified address family.
EMSGSIZE	The socket requires that message be sent atomically, and the message was too long.
ENOMEM	There was insufficient memory available to complete the operation.
ENOSR	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK	<i>fd</i> is not a socket.

SEE ALSO

`libt6(3N)`, `t6set_attr(3N)`, `t6set_endpt_default(3N)`, *Trusted Solaris Developer's Guide*

NOTES

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.

Only `SOCK_STREAM` sockets created in the `AF_INET` address family support out-of-band data.

Information labels (ILs) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any ILs on communications and files from systems running earlier releases as `ADMIN_LOW`.

Objects still have CMW labels, and CMW labels still include the IL component: `IL[SL]`; however, the IL component is fixed at `ADMIN_LOW`.

As a result, Trusted Solaris 7 has the following characteristics:

- ILs do not display in window labels; SLs (Sensitivity Labels) display alone within brackets.
- ILs do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return `ADMIN_LOW`.
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting ILs are always `ADMIN_LOW`, and cannot be set on any objects.
- Options related to information labels in the `label_encodings(4)` file can be ignored:

```
Markings Name= Marks;  
Float Process Information Label;
```

NAME	t6get_attr, t6set_attr – Get security attributes from or set security attributes in the security-attribute buffer handled by a control structure						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> void * t6get_attr(t6attr_id_t attr_type, const t6ctrl_t t6ctl, t6attr_t t6ctl); int t6set_attr(t6attr_id_t attr_type, const void * attr_buf);</pre>						
DESCRIPTION	<p>t6get_attr() takes a control structure, <i>t6ctl</i>, and attribute type, <i>attr_type</i>, and returns a pointer to the requested attribute value (type) from the opaque control structure <i>t6ctl</i>. <i>attr_type</i> contains a number (defined in <tsix/t6attrs.h>) that specifies which type of attribute the caller is interested in getting. Only one type can be specified per call.</p> <p>Returned value by t6get_attr() should be type cast to the standard type that represents the type indicated by <i>attr_type</i>.</p> <p>t6set_attr() replaces the requested attribute value (type) in <i>t6ctl</i> with the value to which <i>attr_buf</i> points. The type of the attribute is specified in <i>attr_type</i> as one of the numbers defined in <tsix/t6attrs.h>.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
RETURN VALUES	<p>Upon successful completion, t6get_attr() returns a pointer to the appropriate value if it exists in the attribute structure. Upon failure, t6get_attr() returns NULL. t6set_attr() returns 0 if the attribute structure can contain the requested attribute; if not, t6set_attr() returns -1 and does not change the attribute structure.</p>						
SEE ALSO							
Trusted Solaris 7 Reference Manual	libt6(3N), t6alloc_blk(3N), t6free_blk(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
NOTES	<p>In the Trusted Solaris environment, t6get_attr() returns values of these types:</p> <table> <tr> <td>au_id_t</td><td>Audit ID</td></tr> </table>	au_id_t	Audit ID				
au_id_t	Audit ID						

<code>auditinfo_t</code>	Audit info
<code>bclear_t</code>	Clearance
<code>bslabel_t</code>	Sensitivity label
<code>gid_t</code>	Effective group ID
<code>gid_t</code>	Supplemental group ID s
<code>pattr_t</code>	Process attributes
<code>priv_set_t</code>	Effective privileges
<code>sid_t</code>	Session ID
<code>pid_t</code>	Process ID
<code>uid_t</code>	Effective user ID

This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1- API -compliant systems.

NAME	t6get_endpt_mask, t6set_endpt_mask, t6get_endpt_default, t6set_endpt_default – Get and set endpoint mask, or get and set endpoint default attributes
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6get_endpt_mask(int fd, t6mask_t * mask); int t6set_endpt_mask(int fd, t6mask_t mask); int t6get_endpt_default(int fd, t6mask_t * mask, t6attr_t attr_ptr); int t6set_endpt_default(int fd, t6mask_t mask, const t6attr_t attr_ptr);</pre>
DESCRIPTION	<p>The security extensions on the communication endpoint include a set of default security attributes that may be applied to outgoing data and an attribute mask that designates which attributes are taken from the endpoint's default attributes and which are taken from the process' effective attributes.</p> <p>By default, data written to an endpoint has associated with it the security attributes of the process that wrote the data. However, a privileged process may change the value of the default attribute mask on an endpoint the process had created, and the endpoint's default attributes.</p> <p>t6get_endpt_mask() allows a process to obtain the current setting of the default attribute mask for the endpoint specified by <i>fd</i> . The attribute mask is returned in the parameter <i>mask</i> .</p> <p>t6set_endpt_mask() allows a process to set the bit values of the default attribute mask for the endpoint specified by <i>fd</i> to the value specified by <i>mask</i> . A bit value of 0 indicates the attribute is taken from the process's effective attributes; and a bit value of 1 indicates the attribute is taken from the endpoint's default attributes.</p> <p>t6get_endpt_default() allows a process to get the current setting of the default attributes of the endpoint specified by <i>fd</i> . <i>mask</i> indicates which attributes are present in the <i>attr_ptr</i> parameter. To access <i>attr_ptr</i> , see t6get_attr(3N) .</p> <p>t6set_endpt_default() allows a process to set the default attributes of the endpoint specified by <i>fd</i> to the attributes specified by <i>attr_ptr</i> . <i>mask</i> indicates which attributes are present in <i>attr_ptr</i> . To set up <i>attr_ptr</i> , see t6set_attr(3N) .</p> <p>Only a process with the appropriate override privileges can change the endpoint's attribute mask or default attributes. To change an endpoint's default attribute or its mask bit, a process must have the override privilege corresponding to the attribute. The override privilege required to specify a default attribute is implementation-specific.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

Upon successful completion, these calls return 0 . If either call encounters an error, the call returns -1 .

SEE ALSO

**Trusted Solaris 7
Reference Manual**

libt6(3N) , t6sendto(3N) , t6set_attr(3N)

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX (RE) 1.1- API -compliant systems.

NAME	t6get_endpt_mask, t6set_endpt_mask, t6get_endpt_default, t6set_endpt_default – Get and set endpoint mask, or get and set endpoint default attributes
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6get_endpt_mask(int fd, t6mask_t * mask); int t6set_endpt_mask(int fd, t6mask_t mask); int t6get_endpt_default(int fd, t6mask_t * mask, t6attr_t attr_ptr); int t6set_endpt_default(int fd, t6mask_t mask, const t6attr_t attr_ptr);</pre>
DESCRIPTION	<p>The security extensions on the communication endpoint include a set of default security attributes that may be applied to outgoing data and an attribute mask that designates which attributes are taken from the endpoint's default attributes and which are taken from the process' effective attributes.</p> <p>By default, data written to an endpoint has associated with it the security attributes of the process that wrote the data. However, a privileged process may change the value of the default attribute mask on an endpoint the process had created, and the endpoint's default attributes.</p> <p>t6get_endpt_mask() allows a process to obtain the current setting of the default attribute mask for the endpoint specified by <i>fd</i> . The attribute mask is returned in the parameter <i>mask</i> .</p> <p>t6set_endpt_mask() allows a process to set the bit values of the default attribute mask for the endpoint specified by <i>fd</i> to the value specified by <i>mask</i> . A bit value of 0 indicates the attribute is taken from the process's effective attributes; and a bit value of 1 indicates the attribute is taken from the endpoint's default attributes.</p> <p>t6get_endpt_default() allows a process to get the current setting of the default attributes of the endpoint specified by <i>fd</i> . <i>mask</i> indicates which attributes are present in the <i>attr_ptr</i> parameter. To access <i>attr_ptr</i> , see t6get_attr(3N) .</p> <p>t6set_endpt_default() allows a process to set the default attributes of the endpoint specified by <i>fd</i> to the attributes specified by <i>attr_ptr</i> . <i>mask</i> indicates which attributes are present in <i>attr_ptr</i> . To set up <i>attr_ptr</i> , see t6set_attr(3N) .</p> <p>Only a process with the appropriate override privileges can change the endpoint's attribute mask or default attributes. To change an endpoint's default attribute or its mask bit, a process must have the override privilege corresponding to the attribute. The override privilege required to specify a default attribute is implementation-specific.</p>
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

Upon successful completion, these calls return 0 . If either call encounters an error, the call returns -1 .

SEE ALSO

**Trusted Solaris 7
Reference Manual**

libt6(3N) , t6sendto(3N) , t6set_attr(3N)

**SunOS 5.7 Reference
Manual**

attributes(5)

NOTES

This man page is based on the version from the TSIX (RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX (RE) 1.1- API -compliant systems.

NAME	t6size_attr – Get the size of a particular attribute from the control structure						
SYNOPSIS	<pre>cc [flags...] file ... -lt6 #include <tsix/t6attrs.h> int t6size_attr(t6attr_id_t attr_type, const t6attr_t t6ctl);</pre>						
RETURN VALUES	<p>t6size_attr() returns the size of an attribute indicated by <i>attr_type</i>.</p> <p>If the t6attr_t control structure <i>t6ctl</i> is a NULL pointer, t6size_attr() returns either the size of a fixed-size attribute or the maximum size of a variable-size attribute. If the <i>attr_type</i> is invalid, t6size_attr() returns 0.</p> <p>If the t6attr_t control structure <i>t6ctl</i> is not NULL, t6size_attr() returns either the size of a fixed-size attribute or the actual size occupied by a variable-size attribute in the control structure <i>t6ctl</i>. If the <i>attr_type</i> is invalid or not in the <i>t6ctl</i>, t6size_attr() returns 0.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> <tr> <td>MT-Level</td><td>MT-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu	MT-Level	MT-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWtsu						
MT-Level	MT-Safe						
SEE ALSO							
Trusted Solaris 7 Reference Manual	t6copy_blk(3N), t6dup_blk(3N)						
SunOS 5.7 Reference Manual	attributes(5)						
NOTES	<p>This man page is based on the version from the TSIX(RE) 1.1 Application Programming Interface (API) document; and this interface is available in TSIX(RE) 1.1-API-compliant systems.</p>						

NAME	t_accept – Accept a connection request
SYNOPSIS	<pre>#include <xti.h> int t_accept(int fd, int resfd, const struct t_call *call);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces that evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, a different header file, <code><tiuser.h></code>, must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is issued by a transport user to accept a connection request. The parameter <i>fd</i> identifies the local transport endpoint where the connection indication arrived; <i>resfd</i> specifies the local transport endpoint where the connection is to be established, and <i>call</i> contains information required by the transport provider to complete the connection. The parameter <i>call</i> points to a <code>t_call</code> structure which contains the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;</pre> <p>In <i>call</i>, <i>addr</i> is the protocol address of the calling transport user, <i>opt</i> indicates any options associated with the connection, <i>udata</i> points to any user data to be returned to the caller, and <i>sequence</i> is the value returned by <code>t_listen(3N)</code> that uniquely associates the response with a previously received connection indication. The address of the caller, <i>addr</i> may be null (length zero). Where <i>addr</i> is not null then it may optionally be checked by XTI.</p> <p>A transport user may accept a connection on either the same, or on a different, local transport endpoint than the one on which the connection indication arrived. Before the connection can be accepted on the same endpoint (<i>resfd</i>==<i>fd</i>), the user must have responded to any previous connection indications received on that transport endpoint by means of <code>t_accept()</code> or <code>t_snddis(3N)</code>. Otherwise, <code>t_accept()</code> will fail and set <code>t_errno</code> to <code>TINDOUT</code>.</p> <p>If a different transport endpoint is specified (<i>resfd</i>!=<i>fd</i>), then the user may or may not choose to bind the endpoint before the <code>t_accept()</code> is issued. If the endpoint is not bound prior to the <code>t_accept()</code>, the endpoint must be in the <code>T_UNBND</code> state before the <code>t_accept()</code> is issued, and the transport provider will automatically bind it to an address that is appropriate for the protocol concerned. If the transport user chooses to bind the endpoint it must be bound to a protocol address with a <i>qlen</i> of zero and must be in the <code>T_IDLE</code> state before the <code>t_accept()</code> is issued.</p>

Responding endpoints should be supplied to `t_accept()` in the state `T_UNBND`.

The call to `t_accept()` may fail with `t_errno` set to `TLOOK` if there are indications (for example connect or disconnect) waiting to be received on endpoint *fd*. Applications should be prepared for such a failure.

The *udata* argument enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider as returned in the *connect* field of the *info* argument of `t_open(3N)` or `t_getinfo(3N)`. If the *len* field of *udata* is zero, no data will be sent to the caller. All the *maxlen* fields are meaningless.

When the user does not indicate any option (*call*→*opt.len* = 0) the connection shall be accepted with the option values currently set for the responding endpoint *resfd*.

RETURN VALUES

`t_accept()` returns:

0 On success.

-1 On failure, and sets `errno` to indicate the error.

VALID STATES

fd: `T_INCON`

resfd (*fd*!=*resfd*): `T_IDLE`, `T_UNBND`

ERRORS

On failure, `t_errno` is set to one of the following:

`TACCES`

The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options.

`TBADADDR`

The specified protocol address was in an incorrect format or contained illegal information.

`TBADDATA`

The amount of user data specified was not within the bounds allowed by the transport provider.

`TBADF`

The file descriptor *fd* or *resfd* does not refer to a transport endpoint.

`TBADOPT`

The specified options were in an incorrect format or contained illegal information.

`TBADSEQ`

Either an invalid sequence number was specified, or a valid sequence number was specified but the connection request was aborted by the peer. In the latter case, its `T_DISCONNECT` event will be received on the listening endpoint.

TINDOUT	The function was called with <i>fd</i> == <i>resfd</i> but there are outstanding connection indications on the endpoint. Those other connection indications must be handled either by rejecting them by means of <code>t_snddis(3N)</code> or accepting them on a different endpoint by means of <code>t_accept()</code> .
TLOOK	An asynchronous event has occurred on the transport endpoint referenced by <i>fd</i> and requires immediate attention.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TOUTSTATE	The communications endpoint referenced by <i>fd</i> or <i>resfd</i> is not in one of the states in which a call to this function is valid.
TPROTO	This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).
TPROVMISMATCH	The file descriptors <i>fd</i> and <i>resfd</i> do not refer to the same transport provider.
TRESADDR	This transport provider requires both <i>fd</i> and <i>resfd</i> to be bound to the same address. This error results if they are not.
TRESQLEN	The endpoint referenced by <i>resfd</i> (where <i>resfd</i> != <i>fd</i>) was bound to a protocol address with a <i>qlen</i> that is greater than zero.
TSYSERR	A system error has occurred during execution of this function.

TLI COMPATIBILITY

Interface Header

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

The XTI interfaces use the header file, `<xti.h>`. TLI interfaces should *not* use this header. They should use the header:

```
#include tiuser.h
```

**Error Description
Values**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO
TINDOUT
TPROVMISMATCH
TRESADDR
TRESQLEN

Option Buffer

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not specify the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the calling process possesses the `PRIV_NET_MAC_READ` privilege and the socket has been bound to a multilevel port (MLP), the connection is accepted on a MLP; otherwise, the connection is accepted on a single-level port (SLP). See `bind(3N)` for more information.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`bind(3N)`, `t_optmgmt(2N)`, `t_bind(3N)`

**SunOS 5.7 Reference
Manual**

`t_connect(3N)`, `t_getinfo(3N)`, `t_getstate(3N)`, `t_listen(3N)`,
`t_open(3N)`, `t_rcvconnect(3N)`, `t_snddis(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

WARNINGS

There may be transport provider-specific restrictions on address binding.

Some transport providers do not differentiate between a connection indication and the connection itself. If the connection has already been established after a successful return of `t_listen(3N)`, `t_accept()` will assign the existing connection to the transport endpoint specified by *resfd*.

NAME	t_bind – Bind an address to a transport endpoint
SYNOPSIS	<pre>#include <xti.h> int t_bind(int fd, const struct t_bind *req, struct t_bind *ret);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces that evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <tiuser.h> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function associates a protocol address with the transport endpoint specified by <i>fd</i> and activates that transport endpoint. In connection mode, the transport provider may begin enqueueing incoming connect indications, or servicing a connection request on the transport endpoint. In connectionless mode, the transport user may send or receive data units through the transport endpoint.</p> <p>The <i>req</i> and <i>ret</i> arguments point to a t_bind structure containing the following members:</p> <pre>struct netbuf addr; unsigned qlen;</pre> <p>The <i>addr</i> field of the t_bind structure specifies a protocol address, and the <i>qlen</i> field is used to indicate the maximum number of outstanding connection indications.</p> <p>The parameter <i>req</i> is used to request that an address, represented by the netbuf structure, be bound to the given transport endpoint. The parameter <i>len</i> specifies the number of bytes in the address, and <i>buf</i> points to the address buffer. The parameter <i>maxlen</i> has no meaning for the <i>req</i> argument. On return, <i>ret</i> contains an encoding for the address that the transport provider actually bound to the transport endpoint; if an address was specified in <i>req</i>, this will be an encoding of the same address. In <i>ret</i>, the user specifies <i>maxlen</i>, which is the maximum size of the address buffer, and <i>buf</i> which points to the buffer where the address is to be placed. On return, <i>len</i> specifies the number of bytes in the bound address, and <i>buf</i> points to the bound address. If <i>maxlen</i> equals zero, no address is returned. If <i>maxlen</i> is greater than zero and less than the length of the address, t_bind() fails with t_errno set to TBUFOVFLW.</p> <p>If the requested address is not available, t_bind() will return -1 with t_errno set as appropriate. If no address is specified in <i>req</i> (the <i>len</i> field of <i>addr</i> in <i>req</i> is zero or <i>req</i> is NULL), the transport provider will assign an appropriate</p>

address to be bound, and will return that address in the *addr* field of *ret*. If the transport provider could not allocate an address, `t_bind()` will fail with `t_errno` set to `TNOADDR`.

The parameter *req* may be a null pointer if the user does not wish to specify an address to be bound. Here, the value of *qlen* is assumed to be zero, and the transport provider will assign an address to the transport endpoint. Similarly, *ret* may be a null pointer if the user does not care what address was bound by the provider and is not interested in the negotiated value of *qlen*. It is valid to set *req* and *ret* to the null pointer for the same call, in which case the provider chooses the address to bind to the transport endpoint and does not return that information to the user.

The *qlen* field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connection indications that the transport provider should support for the given transport endpoint. An outstanding connection indication is one that has been passed to the transport user by the transport provider but which has not been accepted or rejected. A value of *qlen* greater than zero is only meaningful when issued by a passive transport user that expects other users to call it. The value of *qlen* will be negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connection indications. However, this value of *qlen* will never be negotiated from a requested value greater than zero to zero. This is a requirement on transport providers; see `WARNINGS` below. On return, the *qlen* field in *ret* will contain the negotiated value.

If *fd* refers to a connection-mode service, this function allows more than one transport endpoint to be bound to the same protocol address. But it is not possible to bind more than one protocol address to the same transport endpoint. However, the transport provider must also support this capability. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connection indications associated with that protocol address. In other words, only one `t_bind()` for a given protocol address may specify a value of *qlen* greater than zero. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connection indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of *qlen* greater than zero, `t_bind()` will return `-1` and set `t_errno` to `TADDRBUSY`. When a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address will be found to be busy for the duration of the connection, until a `t_unbind(3)` or `t_close(3)` call has been issued. No other transport endpoints may be bound for listening on that same protocol address while that initial listening endpoint is active (in the data transfer phase or in the `T_IDLE` state). This will prevent more than one transport endpoint bound to the same protocol address from accepting connection indications.

If *fd* refers to connectionless mode service, this function allows for more than one transport endpoint to be associated with a protocol address, where the underlying transport provider supports this capability (often in conjunction with value of a protocol-specific option). If a user attempts to bind a second transport endpoint to an already bound protocol address when such capability is not supported for a transport provider, `t_bind()` will return `-1` and set `t_errno` to `TADDRBUSY`.

RETURN VALUES

`t_bind()` returns:

0 On success.

-1 On failure, and sets `t_errno` to indicate the error.

VALID STATES

`T_UNBND`

ERRORS

On failure, `t_errno` is set to one of the following:

`TACCES` The user does not have permission to use the specified address.

`TADDRBUSY` The requested address is in use.

`TBADADDR` The specified protocol address was in an incorrect format or contained illegal information.

`TBADF` The specified file descriptor does not refer to a transport endpoint.

`TBUFOVFLW` The number of bytes allowed for an incoming argument (*maxlen*) is greater than 0 but not sufficient to store the value of that argument. The provider's state will change to `T_IDLE` and the information to be returned in *ret* will be discarded.

`TOUTSTATE` The communications endpoint referenced by *fd* is not in one of the states in which a call to this function is valid.

`TNOADDR` The transport provider could not allocate an address.

`TPROTO` This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (`t_errno`).

`TSYSERR` A system error has occurred during execution of this function.

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include tiuser.h
```

Address Bound

The user can compare the addresses in *req* and *ret* to determine whether the transport provider bound the transport endpoint to a different address than that requested.

Error Description Values

The `t_errno` values `TPROTO` and `TADDRBUSY` can be set by the XTI interface but cannot be set by the TLI interface.

A `t_errno` value that this routine can return under different circumstances than its XTI counterpart is `TBUFOVFLW`. It can be returned even when the `maxlen` field of the corresponding buffer has been set to zero.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

If the calling process possesses the `PRIV_NET_MAC_READ` privilege, the socket is bound to a multilevel port (MLP); otherwise, the connection is bound to a single-level port (SLP). See `bind(3N)` for more information.

SEE ALSO

Trusted Solaris 7
Reference Manual

`bind(3N)`, `t_accept(3N)`

SunOS 5.7 Reference
Manual

`t_alloc(3)`, `t_close(3)`, `t_connect(3)`, `t_unbind(3)`, `attributes(5)`

WARNINGS

The requirement that the value of *qlen* never be negotiated from a requested value greater than zero to zero implies that transport providers, rather than the XTI implementation itself, accept this restriction.

An implementation need not allow an application explicitly to bind more than one communications endpoint to a single protocol address, while permitting more than one connection to be accepted to the same protocol address. That means that although an attempt to bind a communications endpoint to some address with *qlen=0* might be rejected with `TADDRBUSY`, the user may nevertheless use this (unbound) endpoint as a responding endpoint in a call to

t_accept(3N). To become independent of such implementation differences, the user should supply unbound responding endpoints to t_accept(3N).

The local address bound to an endpoint may change as result of a t_accept(3N) or t_connect(3) call. Such changes are not necessarily reversed when the connection is released.

NAME	t_optmgmt – Manage options for a transport endpoint
SYNOPSIS	<pre>#include <xti.h> int t_optmgmt(int fd, const struct t_optmgmt *req, struct t_optmgmt *ret);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <tiuser.h> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>The t_optmgmt() function enables a transport user to retrieve, verify or negotiate protocol options with the transport provider. The argument <i>fd</i> identifies a transport endpoint.</p> <p>The <i>req</i> and <i>ret</i> arguments point to a t_optmgmt structure containing the following members:</p> <pre>struct netbuf opt; t_scalar_t flags;</pre> <p>The <i>opt</i> field identifies protocol options and the <i>flags</i> field is used to specify the action to take with those options.</p> <p>The options are represented by a netbuf structure in a manner similar to the address in t_bind(3N). The argument <i>req</i> is used to request a specific action of the provider and to send options to the provider. The argument <i>len</i> specifies the number of bytes in the options, <i>buf</i> points to the options buffer, and <i>maxlen</i> has no meaning for the <i>req</i> argument. The transport provider may return options and flag values to the user through <i>ret</i>. For <i>ret</i>, <i>maxlen</i> specifies the maximum size of the options buffer and <i>buf</i> points to the buffer where the options are to be placed. If <i>maxlen</i> in <i>ret</i> is set to zero, no options values are returned. On return, <i>len</i> specifies the number of bytes of options returned. The value in <i>maxlen</i> has no meaning for the <i>req</i> argument, but must be set in the <i>ret</i> argument to specify the maximum number of bytes the options buffer can hold.</p> <p>Each option in the options buffer is of the form struct t_opthdr possibly followed by an option value.</p> <p>The <i>level</i> field of struct t_opthdr identifies the XTI level or a protocol of the transport provider. The <i>name</i> field identifies the option within the level, and <i>len</i> contains its total length; that is, the length of the option header t_opthdr plus the length of the option value. If t_optmgmt() is called with the action T_NEGOTIATE set, the <i>status</i> field of the returned options contains information about the success or failure of a negotiation.</p>

Several options can be concatenated. The option user has, however to ensure that each options header and value part starts at a boundary appropriate for the architecture-specific alignment rules. The macros `T_OPT_FIRSTHDR(nbp)`, `T_OPT_NEXTHDR(nbp, tohp)`, `T_OPT_DATA(tohp)` are provided for that purpose.

`T_OPT_DATA(nhp)`

If argument is a pointer to a `t_opthdr` structure, this macro returns an unsigned character pointer to the data associated with the `t_opthdr`.

`T_OPT_NEXTHDR(nbp, tohp)`

If the first argument is a pointer to a `netbuf` structure associated with an option buffer and second argument is a pointer to a `t_opthdr` structure within that option buffer, this macro returns a pointer to the next `t_opthdr` structure or a null pointer if this `t_opthdr` is the last `t_opthdr` in the option buffer.

`T_OPT_FIRSTHDR(nbp)`

If the argument is a pointer to a `netbuf` structure associated with an option buffer, this macro returns the pointer to the first `t_opthdr` structure in the associated option buffer, or a null pointer if there is no option buffer associated with this `netbuf` or if it is not possible or the associated option buffer is too small to accommodate even the first aligned option header.

`T_OPT_FIRSTHDR` is useful for finding an appropriately aligned start of the option buffer. `T_OPT_NEXTHDR` is useful for moving to the start of the next appropriately aligned option in the option buffer. Note that `T_OPT_NEXTHDR` is also available for backward compatibility requirements. `T_OPT_DATA` is useful for finding the start of the data part in the option buffer where the contents of its values start on an appropriately aligned boundary.

If the transport user specifies several options on input, all options must address the same level.

If any option in the options buffer does not indicate the same level as the first option, or the level specified is unsupported, then the `t_optmgmt()` request will fail with `TBADOPT`. If the error is detected, some options have possibly been successfully negotiated. The transport user can check the current status by calling `t_optmgmt()` with the `T_CURRENT` flag set.

The *flags* field of *req* must specify one of the following actions:

`T_NEGOTIATE`

This action enables the transport user to negotiate option values.

The user specifies the options of interest and their values in the buffer specified by *req*→*opt.buf* and *req*→*opt.len*. The negotiated option values are

returned in the buffer pointed to by *ret->opt.buf*. The *status* field of each returned option is set to indicate the result of the negotiation. The value is `T_SUCCESS` if the proposed value was negotiated, `T_PARTSUCCESS` if a degraded value was negotiated, `T_FAILURE` if the negotiation failed (according to the negotiation rules), `T_NOTSUPPORT` if the transport provider does not support this option or illegally requests negotiation of a privileged option, and `T_READONLY` if modification of a read-only option was requested. If the status is `T_SUCCESS`, `T_FAILURE`, `T_NOTSUPPORT` or `T_READONLY`, the returned option value is the same as the one requested on input.

The overall result of the negotiation is returned in *ret->flags*.

This field contains the worst single result, whereby the rating is done according to the order `T_NOTSUPPORT`, `T_READONLY`, `T_FAILURE`, `T_PARTSUCCESS`, `T_SUCCESS`. The value `T_NOTSUPPORT` is the worst result and `T_SUCCESS` is the best.

For each level, the option `T_ALLOPT` can be requested on input. No value is given with this option; only the `t_opthdr` part is specified. This input requests to negotiate all supported options of this level to their default values. The result is returned option by option in *ret->opt.buf*. Note that depending on the state of the transport endpoint, not all requests to negotiate the default value may be successful.

`T_CHECK`

This action enables the user to verify whether the options specified in *req* are supported by the transport provider. If an option is specified with no option value (it consists only of a `t_opthdr` structure), the option is returned with its *status* field set to `T_SUCCESS` if it is supported, `T_NOTSUPPORT` if it is not or needs additional user privileges, and `T_READONLY` if it is read-only (in the current XTI state). No option value is returned.

If an option is specified with an option value, the *status* field of the returned option has the same value, as if the user had tried to negotiate this value with `T_NEGOTIATE`. If the status is `T_SUCCESS`, `T_FAILURE`, `T_NOTSUPPORT` or `T_READONLY`, the returned option value is the same as the one requested on input.

The overall result of the option checks is returned in *ret->flags*. This field contains the worst single result of the option checks, whereby the rating is the same as for `T_NEGOTIATE`.

Note that no negotiation takes place. All currently effective option values remain unchanged.

`T_DEFAULT`

This action enables the transport user to retrieve the default option values. The user specifies the options of interest in *req*→*opt.buf*. The option values are irrelevant and will be ignored; it is sufficient to specify the *t_opthdr* part of an option only. The default values are then returned in *ret*→*opt.buf*.

The *status* field returned is *T_NOTSUPPORT* if the protocol level does not support this option or the transport user illegally requested a privileged option, *T_READONLY* if the option is read-only, and set to *T_SUCCESS* in all other cases. The overall result of the request is returned in *ret*→*flags*. This field contains the worst single result, whereby the rating is the same as for *T_NEGOTIATE*.

For each level, the option *T_ALLOPT* can be requested on input. All supported options of this level with their default values are then returned. In this case, *ret*→*opt.maxlen* must be given at least the value *info*→*options* before the call. See *t_getinfo*(3N) and *t_open*(3N).

T_CURRENT

This action enables the transport user to retrieve the currently effective option values. The user specifies the options of interest in *req*→*opt.buf*. The option values are irrelevant and will be ignored; it is sufficient to specify the *t_opthdr* part of an option only. The currently effective values are then returned in *req*→*opt.buf*.

The *status* field returned is *T_NOTSUPPORT* if the protocol level does not support this option or the transport user illegally requested a privileged option, *T_READONLY* if the option is read-only, and set to *T_SUCCESS* in all other cases. The overall result of the request is returned in *ret*→*flags*. This field contains the worst single result, whereby the rating is the same as for *T_NEGOTIATE*.

For each level, the option *T_ALLOPT* can be requested on input. All supported options of this level with their currently effective values are then returned.

The option *T_ALLOPT* can only be used with *t_optmgmt*() and the actions *T_NEGOTIATE*, *T_DEFAULT* and *T_CURRENT*. It can be used with any supported level and addresses all supported options of this level. The option has no value; it consists of a *t_opthdr* only. Since in a *t_optmgmt*() call only options of one level may be addressed, this option should not be requested together with other options. The function returns as soon as this option has been processed.

Options are independently processed in the order they appear in the input option buffer. If an option is multiply input, it depends on the implementation whether it is multiply output or whether it is returned only once.

Transport providers may not be able to provide an interface capable of supporting `T_NEGOTIATE` and/or `T_CHECK` functionalities. When this is the case, the error `TNOTSUPPORT` is returned.

The function `t_optmgmt()` may block under various circumstances and depending on the implementation. The function will block, for instance, if the protocol addressed by the call resides on a separate controller. It may also block due to flow control constraints; that is, if data sent previously across this transport endpoint has not yet been fully processed. If the function is interrupted by a signal, the option negotiations that have been done so far may remain valid. The behavior of the function is not changed if `O_NONBLOCK` is set.

RETURN VALUES

`t_optmgmt()` returns:

0 On success.

-1 On failure, and sets `t_errno` to indicate the error.

VALID STATES

ALL - apart from `T_UNINIT`.

ERRORS

On failure, `t_errno` is set to one of the following:

`TBADF` The specified file descriptor does not refer to a transport endpoint.

`TBADFLAG` An invalid flag was specified.

`TBADOPT` The specified options were in an incorrect format or contained illegal information.

`TBUFOVFLW` The number of bytes allowed for an incoming argument (*maxlen*) is greater than 0 but not sufficient to store the value of that argument. The information to be returned in *ret* will be discarded.

`TNOTSUPPORT` This action is not supported by the transport provider.

`TOUTSTATE` The communications endpoint referenced by *fd* is not in one of the states in which a call to this function is valid.

`TPROTO` This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (`t_errno`).

`TSYSERR` A system error has occurred during execution of this function, or the specified option requires `PRIV_NET_RAWACCESS` privilege.

TLI COMPATIBILITY	The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.
Interface Header	<p>The XTI interfaces use the header file, <code>xti.h</code>. TLI interfaces should <i>not</i> use this header. They should use the header:</p> <pre>#include tiuser.h</pre>
Error Description Values	<p>The <code>t_errno</code> value <code>TPROTO</code> can be set by the XTI interface but not by the TLI interface.</p> <p>The <code>t_errno</code> values that this routine can return under different circumstances than its XTI counterpart are <code>TACCES</code> and <code>TBUFOVFLW</code>.</p> <p><code>TACCES</code> can be returned to indicate that the user does not have permission to negotiate the specified options.</p> <p><code>TBUFOVFLW</code> can be returned even when the <code>maxlen</code> field of the corresponding buffer has been set to zero.</p>
Option Buffers	The format of the options in an <code>opt</code> buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format. The macros <code>T_OPT_DATA</code> , <code>T_OPT_NEXTHDR</code> , and <code>T_OPT_FIRSTHDR</code> described for XTI are not available for use by TLI interfaces.
Actions	<p>The semantic meaning of various action values for the <i>flags</i> field of <i>req</i> differs between the TLI and XTI interfaces. TLI interface users should heed the following descriptions of the actions:</p> <p><code>T_NEGOTIATE</code> This action enables the user to negotiate the values of the options specified in <i>req</i> with the transport provider. The provider will evaluate the requested options and negotiate the values, returning the negotiated values through <i>ret</i>.</p> <p><code>T_CHECK</code> This action enables the user to verify whether the options specified in <i>req</i> are supported by the transport provider. On return, the <i>flags</i> field of <i>ret</i> will have either <code>T_SUCCESS</code> or <code>T_FAILURE</code> set to indicate to the user whether the options are supported. These flags are only meaningful for the <code>T_CHECK</code> request.</p> <p><code>T_DEFAULT</code> This action enables a user to retrieve the default options supported by the transport provider into the <code>opt</code> field of <i>ret</i>. In <i>req</i>, the <code>len</code> field of <code>opt</code> must be zero and the <code>buf</code> field may be <code>NULL</code>.</p>

Connectionless-Mode

If issued as part of the connectionless-mode service, `t_optmgmt()` may block due to flow control constraints. The function will not complete until the transport provider has processed all previously sent data units.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

A process must have the `PRIV_NET_RAWACCESS` privilege in order to specify IP options 130 or 134. The former refers to the RIPS0 Basic Security Option and the later refers to the CIPSO option.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`t_accept(3N)`, `t_bind(3N)`

**SunOS 5.7 Reference
Manual**

`close(2)`, `poll(2)`, `select(3C)`, `t_alloc(3N)`, `t_close(3N)`,
`t_connect(3N)`, `t_getinfo(3N)`, `t_listen(3N)`, `t_open(3N)`, `t_rcv(3N)`,
`t_rcvconnect(3N)`, `t_rcvudata(3N)`, `t_snddis(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

NAME	t_snd – Send data or expedited data over a connection
SYNOPSIS	<pre>#include <xti.h> int t_snd(int fd, void *buf, unsigned int nbytes, int flags);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <tiuser.h> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used to send either normal or expedited data. The argument <i>fd</i> identifies the local transport endpoint over which data should be sent, <i>buf</i> points to the user data, <i>nbytes</i> specifies the number of bytes of user data to be sent, and <i>flags</i> specifies any optional flags described below:</p> <p>T_EXPEDITED If set in <i>flags</i>, the data will be sent as expedited data and will be subject to the interpretations of the transport provider.</p> <p>T_MORE If set in <i>flags</i>, this indicates to the transport provider that the transport service data unit (TSDU) (or expedited transport service data unit - ETSDU) is being sent through multiple <code>t_snd()</code> calls. Each <code>t_snd()</code> with the <code>T_MORE</code> flag set indicates that another <code>t_snd()</code> will follow with more data for the current TSDU (or ETSDU).</p> <p>The end of the TSDU (or ETSDU) is identified by a <code>t_snd()</code> call with the <code>T_MORE</code> flag not set. Use of <code>T_MORE</code> enables a user to break up large logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the <i>info</i> argument on return from <code>t_open(3N)</code> or <code>t_getinfo(3N)</code>, the <code>T_MORE</code> flag is not meaningful and will be ignored if set.</p> <p>The sending of a zero-length fragment of a TSDU or ETSDU is only permitted where this is used to indicate the end of a TSDU or ETSDU; that is, when the <code>T_MORE</code> flag is not set. Some transport providers also forbid zero-length TSDUs and ETSDUs.</p> <p>T_PUSH If set in <i>flags</i>, requests that the provider transmit all data that it has accumulated but not sent. The request is a local action on the provider and does not affect any similarly</p>

named protocol flag (for example, the TCP PUSH flag). This effect of setting this flag is protocol-dependent, and it may be ignored entirely by transport providers which do not support the use of this feature.

Note that the communications provider is free to collect data in a send buffer until it accumulates a sufficient amount for transmission.

By default, `t_snd()` operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if `O_NONBLOCK` is set by means of `t_open(3N)` or `fcntl(2)`, `t_snd()` will execute in asynchronous mode, and will fail immediately if there are flow control restrictions. The process can arrange to be informed when the flow control restrictions are cleared by means of either `t_look(3N)` or the EM interface.

On successful completion, `t_snd()` returns the number of bytes (octets) accepted by the communications provider. Normally this will equal the number of octets specified in `nbytes`. However, if `O_NONBLOCK` is set or the function is interrupted by a signal, it is possible that only part of the data has actually been accepted by the communications provider. In this case, `t_snd()` returns a value that is less than the value of `nbytes`. If `t_snd()` is interrupted by a signal before it could transfer data to the communications provider, it returns `-1` with `t_errno` set to `TSYSERR` and `errno` set to `EINTR`.

If `nbytes` is zero and sending of zero bytes is not supported by the underlying communications service, `t_snd()` returns `-1` with `t_errno` set to `TBADDATA`.

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as specified by the current values in the TSDU or ETSDU fields in the *info* argument returned by `t_getinfo(3N)`.

The error `TLOOK` is returned for asynchronous events. It is required only for an incoming disconnect event but may be returned for other events.

RETURN VALUES

On successful completion, `t_snd()` returns the number of bytes accepted by the transport provider. Otherwise, `-1` is returned on failure and `t_errno` is set to indicate the error.

Note that if the number of bytes accepted by the communications provider is less than the number of bytes requested, this may either indicate that `O_NONBLOCK` is set and the communications provider is blocked due to flow control, or that `O_NONBLOCK` is clear and the function was interrupted by a signal.

ERRORS

On failure, `t_errno` is set to one of the following:

`TBADDATA` Illegal amount of data:

- A single send was attempted specifying a TSDU (ETSDU) or fragment TSDU (ETSDU) greater than that specified by the current values of the TSDU or ETSDU fields in the *info* argument.
- A send of a zero byte TSDU (ETSDU) or zero byte fragment of a TSDU (ETSDU) is not supported by the provider.
- Multiple sends were attempted resulting in a TSDU (ETSDU) larger than that specified by the current value of the TSDU or ETSDU fields in the *info* argument – the ability of an XTI implementation to detect such an error case is implementation-dependent. See WARNINGS, below.

TBADF	The specified file descriptor does not refer to a transport endpoint.
TBADFLAG	An invalid flag was specified.
TFLOW	O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.
TLOOK	An asynchronous event has occurred on this transport endpoint.
TNOTSUPPORT	This function is not supported by the underlying transport provider.
TOUTSTATE	The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.
TPROTO	This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<i>t_errno</i>).
TSYSERR	A system error has occurred during execution of this function.

TLI COMPATIBILITY

Interface Header

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include tiuser.h
```

**Error Description
Values**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

TPROTO
TLOOK
TBADFLAG
TOUTSTATE

The `t_errno` value that this routine can return under different circumstances than its XTI counterpart is `TBADDATA`.

In the case of a `TBADDATA` error, `TBADDATA` is returned, only for illegal zero byte TSDU (ETSDU) send attempts.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the process calling these routines possesses the `PRIV_NET_REPLY_EQUAL` privilege, the packets that the process sends will carry the same CMW label and clearance as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.

SEE ALSO

Trusted Solaris 7
Reference Manual

`fcntl(2)`

SunOS 5.7 Reference
Manual

`t_getinfo(3N)`, `t_look(3N)`, `t_open(3N)`, `t_t_rcv(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

WARNINGS

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. Therefore if several processes issue concurrent `t_snd()` calls then the different data may be intermixed.

Multiple sends which exceed the maximum TSDU or ETSDU size may not be discovered by XTI. In this case an implementation-dependent error will result, generated by the transport provider, perhaps on a subsequent XTI call. This error may take the form of a connection abort, a `TSYSERR`, a `TBADDATA` or a `TPROTO` error.

If multiple sends which exceed the maximum TSDU or ETSDU size are detected by XTI, `t_snd()` fails with `TBADDATA`.

NAME	t_sndudata – Send a data unit
SYNOPSIS	<pre>#include <xti.h> int t_sndudata(int fd, const struct t_unitdata *unitdata);</pre>
DESCRIPTION	<p>This routine is part of the XTI interfaces which evolved from the TLI interfaces. XTI represents the future evolution of these interfaces. However, TLI interfaces are supported for compatibility. When using a TLI routine that has the same name as an XTI routine, the <tiuser.h> header file must be used. Refer to the TLI COMPATIBILITY section for a description of differences between the two interfaces.</p> <p>This function is used in connectionless-mode to send a data unit to another transport user. The argument <i>fd</i> identifies the local transport endpoint through which data will be sent, and <i>unitdata</i> points to a <i>t_unitdata</i> structure containing the following members:</p> <pre>struct netbuf addr; struct netbuf opt; struct netbuf udata;</pre> <p>In <i>unitdata</i>, <i>addr</i> specifies the protocol address of the destination user, <i>opt</i> identifies options that the user wants associated with this request, and <i>udata</i> specifies the user data to be sent. The user may choose not to specify what protocol options are associated with the transfer by setting the <i>len</i> field of <i>opt</i> to zero. In this case, the provider uses the option values currently set for the communications endpoint.</p> <p>If the <i>len</i> field of <i>udata</i> is zero, and sending of zero octets is not supported by the underlying transport service, the <i>t_sndudata()</i> will return set to TBADDDATA.</p> <p>By default, <i>t_sndudata()</i> operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if <i>O_NONBLOCK</i> is set by means of <i>t_open(3N)</i> or <i>fcntl(2)</i>, <i>t_sndudata()</i> will execute in asynchronous mode and will fail under such conditions. The process can arrange to be notified of the clearance of a flow control restriction by means of either <i>t_look(3N)</i> or the EM interface.</p> <p>If the amount of data specified in <i>udata</i> exceeds the TSDU size as returned in the <i>tsdu</i> field of the <i>info</i> argument of <i>t_open(3N)</i> or <i>t_getinfo(3N)</i>, a TBADDDATA error will be generated. If <i>t_sndudata()</i> is called before the destination user has activated its transport endpoint (see <i>t_bind(3N)</i>), the data unit may be discarded.</p>

If it is not possible for the transport provider to immediately detect the conditions that cause the errors `TBADADDR` and `TBADOPT`, these errors will alternatively be returned by `t_rcvuderr`. Therefore, an application must be prepared to receive these errors in both of these ways.

If the call is interrupted, `t_sndudata()` will return `EINTR` and the datagram will not be sent.

RETURN VALUES

`t_sndudata()` returns:

0 On success.

-1 On failure, and sets `errno` to indicate the error.

VALID STATES

`T_IDLE`.

ERRORS

On failure, `t_errno` is set to one of the following:

<code>TBADADDR</code>	The specified protocol address was in an incorrect format or contained illegal information.
<code>TBADDATA</code>	Illegal amount of data. A single send was attempted specifying a TSDU greater than that specified in the <i>info</i> argument, or a send of a zero byte TSDU is not supported by the provider.
<code>TBADF</code>	The specified file descriptor does not refer to a transport endpoint.
<code>TBADOPT</code>	The specified options were in an incorrect format or contained illegal information.
<code>TFLOW</code>	<code>O_NONBLOCK</code> was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.
<code>TLOOK</code>	An asynchronous event has occurred on this transport endpoint.
<code>TNOTSUPPORT</code>	This function is not supported by the underlying transport provider.
<code>TOUTSTATE</code>	The communications endpoint referenced by <i>fd</i> is not in one of the states in which a call to this function is valid.
<code>TPROTO</code>	This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (<code>t_errno</code>).
<code>TSYSERR</code>	A system error has occurred during execution of this function.

**TLI
COMPATIBILITY**

The XTI and TLI interface definitions have common names but use different header files. This, and other semantic differences between the two interfaces are described in the subsections below.

Interface Header

The XTI interfaces use the header file, `xti.h`. TLI interfaces should *not* use this header. They should use the header:

```
#include tiuser.h
```

**Error Description
Values**

The `t_errno` values that can be set by the XTI interface and cannot be set by the TLI interface are:

```
TPROTO
TPROTO
TBADADDR
TBADOPT
TLOOK
TOUTSTATE
```

Notes

Whenever this function fails with `t_error` set to `TFLOW`, `O_NONBLOCK` must have been set.

Option Buffers

The format of the options in an `opt` buffer is dictated by the transport provider. Unlike the XTI interface, the TLI interface does not fix the buffer format.

For more information refer to the *Transport Interfaces Programming Guide*

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

If the process calling these routines possesses the `PRIV_NET_REPLY_EQUAL` privilege, the packets that the process sends will carry the same CMW label and clearance as that of the last packet received from the destination. If no packet from the destination has ever been received, this privilege has no effect.

SEE ALSO

Trusted Solaris 7
Reference Manual

`fcntl(2)`, `t_bind(3N)`

**SunOS 5.7 Reference
Manual**

`t_alloc(3N)`, `t_error(3N)`, `t_getinfo(3N)`, `t_look(3N)`, `t_open(3N)`,
`t_rcvudata(3N)`, `t_rcvuderr(3N)`, `attributes(5)`

Transport Interfaces Programming Guide

NAME	labelbuilder, tsol_lbuild_create, tsol_lbuild_get, tsol_lbuild_set, tsol_lbuild_destroy – Create a Motif-based user interface for interactively building a valid label or clearance
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...] #include <Dt/ModLabel.h> ModLabelData * tsol_lbuild_create(Widget widget void (*event_handler) () ok_callback lbuild_attributes extended_operation, , NULL); void *tsol_lbuild_get(ModLabelData * data, lbuild_attributes extended_operation); void tsol_lbuild_set(ModLabelData * data lbuild_attributes extended_operation, , NULL); void tsol_lbuild_destroy(ModLabelData * data);</pre>
DESCRIPTION	<p>The Label builder user interface prompts the end user for information and generates a valid CMW label, information label, sensitivity label, or clearance from the user input based on specifications in the <code>label_encodings(4)</code> file on the system where the application runs. The end user can build the label or clearance by typing a text value or by interactively choosing options.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, Label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>If the end user does not have the authorization to upgrade or downgrade labels, or if the user-built label is out of the user's accreditation range, the OK and Reset pushbuttons are grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> creates the graphical user interface and returns a pointer variable of type <code>ModLabeldata*</code> that contains information on the user interface. This information is a combination of values passed in the <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <p><code>lbuild_dialog</code> The Label builder dialog box.</p>

ok The OK pushbutton.

cancel The Cancel pushbutton.

reset The Reset pushbutton.

help The Help pushbutton.

The `tsol_lbuild_create()` parameter list takes the following values:

widget The widget from which the dialog box is created. Any Motif widget can be passed.

ok_callback A callback function that implements the behavior of the OK pushbutton on the dialog box.

..., NULL A NULL terminated list of extended operations and value pairs that define the characteristics and behavior of the Label builder dialog box.

`tsol_lbuild_destroy()` destroys the `ModLabelData` structure returned by `tsol_lbuild_create()`.

`tsol_lbuild_get()` and `tsol_lbuild_set()` access the information stored in the `ModLabelData` structure returned by `tsol_lbuild_create()`.

The following extended operations can be passed to `tsol_lbuild_create()` to build the user interface, to `tsol_lbuild_get()` to retrieve information on the user interface, and to `tsol_lbuild_set()` to change the user interface information. All extended operations are valid for `tsol_lbuild_get()`, but the `*WORK*` operations are not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` because these values are set from input supplied by the end user. These exceptions are noted in the descriptions.

LBUILD_MODE

Create a user interface to build an information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

`LBUILD_MODE_SL` Build a sensitivity label.

`LBUILD_MODE_CMW` Build a CMW label.

`LBUILD_MODE_CLR` Build a clearance.

LBUILD_VALUE_SL

The starting sensitivity label. This value is `ADMIN_LOW` by default and is used when the mode is `LBUILD_MODE_SL`.

LBUILD_VALUE_IL

The starting information label. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_IL .

LBUILD_VALUE_CMW

The starting CMW label. This value is ADMIN_LOW[ADMIN_LOW] by default and is used when the mode is LBUILD_MODE_CMW .

LBUILD_VALUE_CLR

The starting clearance. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_CLR .

LBUILD_USERFIELD

A character string prompt that displays at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_SHOW

Show or hide the Label builder dialog box. Value is FALSE by default.

TRUE Show the Label builder dialog box.

FALSE Hide the Label builder dialog box.

LBUILD_TITLE

A character string title that appears at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_WORK_SL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The sensitivity label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_IL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The information label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CMW

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The CMW label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CLR

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The clearance the end user is building. Value is updated to the end user's input

when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_X

The X position in pixels of the top-left corner of the Label buider dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_Y

The Y position in pixels of the top-left corner of the Label builder dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_LOWER_BOUND

The lowest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. This value is the user's minimum label.

LBUILD_UPPER_BOUND

The highest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. A supplied value should be within the user's accreditation range. If no value is specified, the value is the user's workspace sensitivity label, or if the executable has the `PRIV_SYS_TRANS_LABEL` privilege, the value is the user's clearance.

LBUILD_CHECK_AR

Check that the user-built label entered in the Update With field is within the user's accreditation range. A value of 1 means check, and a value of 0 means do not check. If checking is on and the label is out of range, an error message is raised to the end user.

LBUILD_VIEW

Use the internal or external label representation. Value is `LBUILD_VIEW_EXTERNAL` by default.

LBUILD_VIEW_INTERNAL Use the internal names for the highest and lowest labels in the system: `ADMIN_HIGH` and `ADMIN_LOW`.

LBUILD_VIEW_EXTERNAL Promote an `ADMIN_LOW` label to the next highest label, and demote an `ADMIN_HIGH` label to the next lowest label.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The `tsol_lbuild_get()` returns -1 if it is unable to get the value.

The `tsol_lbuild_create()` routine returns a variable of type `ModLabelData` that contains the information provided in the `tsol_lbuild_create()` input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface.

EXAMPLES**EXAMPLE 1** To create a Label Builder

```
(ModLabelData *)lbldata = tsol_lbuild_create(widget0, callback_function,
    LBUILD_MODE, LBUILD_MODE_CMW,
    LBUILD_TITLE, "Setting CMW Label",
    LBUILD_VIEW, LBUILD_VIEW_INTERNAL,
    LBUILD_X, 200,
    LBUILD_Y, 200,
    LBUILD_USERFIELD, "Pathname:",
    LBUILD_SHOW, FALSE,
    NULL);
```

EXAMPLE 2 To query the mode and display the Label Builder

These examples call the `tsol_lbuild_get()` routine to query the mode being used, and call the `tsol_lbuild_set()` routine so the Label builder dialog box displays.

```
mode = (int)tsol_lbuild_get(lbldata, LBUILD_MODE );

tsol_lbuild_set(lbldata, LBUILD_SHOW, TRUE,
    NULL);
```

EXAMPLE 3 To destroy the `ModLabelData` variable

This example destroys the `ModLabelData` variable returned in the call to `tsol_lbuild_create()`.

```
tsol_lbuild_destroy(lbldata);
```

FILES

```
</usr/dt/include/Dt/ModLabel.h>
```

Header file for label builder 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`label_encodings(4)`

Trusted Solaris Developer's Guide *Trusted Solaris administrator's document set*
Trusted Solaris Label Administration *Trusted Solaris User's Guide*

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	labelbuilder, tsol_lbuild_create, tsol_lbuild_get, tsol_lbuild_set, tsol_lbuild_destroy – Create a Motif-based user interface for interactively building a valid label or clearance
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...] #include <Dt/ModLabel.h> ModLabelData * tsol_lbuild_create(Widget widget void (*event_handler) () ok_callback lbuild_attributes extended_operation, , NULL); void *tsol_lbuild_get(ModLabelData * data, lbuild_attributes extended_operation); void tsol_lbuild_set(ModLabelData * data lbuild_attributes extended_operation, , NULL); void tsol_lbuild_destroy(ModLabelData * data);</pre>
DESCRIPTION	<p>The Label builder user interface prompts the end user for information and generates a valid CMW label, information label, sensitivity label, or clearance from the user input based on specifications in the <code>label_encodings(4)</code> file on the system where the application runs. The end user can build the label or clearance by typing a text value or by interactively choosing options.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, Label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>If the end user does not have the authorization to upgrade or downgrade labels, or if the user-built label is out of the user's accreditation range, the OK and Reset pushbuttons are grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> creates the graphical user interface and returns a pointer variable of type <code>ModLabeldata*</code> that contains information on the user interface. This information is a combination of values passed in the <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <p><code>lbuild_dialog</code> The Label builder dialog box.</p>

ok The OK pushbutton.

cancel The Cancel pushbutton.

reset The Reset pushbutton.

help The Help pushbutton.

The `tsol_lbuild_create()` parameter list takes the following values:

widget The widget from which the dialog box is created. Any Motif widget can be passed.

ok_callback A callback function that implements the behavior of the OK pushbutton on the dialog box.

..., NULL A NULL terminated list of extended operations and value pairs that define the characteristics and behavior of the Label builder dialog box.

`tsol_lbuild_destroy()` destroys the `ModLabelData` structure returned by `tsol_lbuild_create()`.

`tsol_lbuild_get()` and `tsol_lbuild_set()` access the information stored in the `ModLabelData` structure returned by `tsol_lbuild_create()`.

The following extended operations can be passed to `tsol_lbuild_create()` to build the user interface, to `tsol_lbuild_get()` to retrieve information on the user interface, and to `tsol_lbuild_set()` to change the user interface information. All extended operations are valid for `tsol_lbuild_get()`, but the `*WORK*` operations are not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` because these values are set from input supplied by the end user. These exceptions are noted in the descriptions.

LBUILD_MODE

Create a user interface to build an information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

LBUILD_MODE_SL Build a sensitivity label.

LBUILD_MODE_CMW Build a CMW label.

LBUILD_MODE_CLR Build a clearance.

LBUILD_VALUE_SL

The starting sensitivity label. This value is `ADMIN_LOW` by default and is used when the mode is `LBUILD_MODE_SL`.

LBUILD_VALUE_IL

The starting information label. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_IL .

LBUILD_VALUE_CMW

The starting CMW label. This value is ADMIN_LOW[ADMIN_LOW] by default and is used when the mode is LBUILD_MODE_CMW .

LBUILD_VALUE_CLR

The starting clearance. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_CLR .

LBUILD_USERFIELD

A character string prompt that displays at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_SHOW

Show or hide the Label builder dialog box. Value is FALSE by default.

TRUE

Show the Label builder dialog box.

FALSE

Hide the Label builder dialog box.

LBUILD_TITLE

A character string title that appears at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_WORK_SL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The sensitivity label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_IL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The information label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CMW

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The CMW label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CLR

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The clearance the end user is building. Value is updated to the end user's input

when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_X

The X position in pixels of the top-left corner of the Label buider dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_Y

The Y position in pixels of the top-left corner of the Label builder dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_LOWER_BOUND

The lowest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. This value is the user's minimum label.

LBUILD_UPPER_BOUND

The highest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. A supplied value should be within the user's accreditation range. If no value is specified, the value is the user's workspace sensitivity label, or if the executable has the `PRIV_SYS_TRANS_LABEL` privilege, the value is the user's clearance.

LBUILD_CHECK_AR

Check that the user-built label entered in the Update With field is within the user's accreditation range. A value of 1 means check, and a value of 0 means do not check. If checking is on and the label is out of range, an error message is raised to the end user.

LBUILD_VIEW

Use the internal or external label representation. Value is `LBUILD_VIEW_EXTERNAL` by default.

LBUILD_VIEW_INTERNAL Use the internal names for the highest and lowest labels in the system: `ADMIN_HIGH` and `ADMIN_LOW` .

LBUILD_VIEW_EXTERNAL Promote an `ADMIN_LOW` label to the next highest label, and demote an `ADMIN_HIGH` label to the next lowest label.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The `tsol_lbuild_get()` returns -1 if it is unable to get the value.

The `tsol_lbuild_create()` routine returns a variable of type `ModLabelData` that contains the information provided in the `tsol_lbuild_create()` input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface.

EXAMPLES

EXAMPLE 1 To create a Label Builder

```
(ModLabelData *)lbldata = tsol_lbuild_create(widget0, callback_function,
    LBUILD_MODE, LBUILD_MODE_CMW,
    LBUILD_TITLE, "Setting CMW Label",
    LBUILD_VIEW, LBUILD_VIEW_INTERNAL,
    LBUILD_X, 200,
    LBUILD_Y, 200,
    LBUILD_USERFIELD, "Pathname:",
    LBUILD_SHOW, FALSE,
    NULL);
```

EXAMPLE 2 To query the mode and display the Label Builder

These examples call the `tsol_lbuild_get()` routine to query the mode being used, and call the `tsol_lbuild_set()` routine so the Label builder dialog box displays.

```
mode = (int)tsol_lbuild_get(lbldata, LBUILD_MODE );

tsol_lbuild_set(lbldata, LBUILD_SHOW, TRUE,
    NULL);
```

EXAMPLE 3 To destroy the `ModLabelData` variable

This example destroys the `ModLabelData` variable returned in the call to `tsol_lbuild_create()`.

```
tsol_lbuild_destroy(lbldata);
```

FILES

`</usr/dt/include/Dt/ModLabel.h>`
Header file for label builder 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.

SEE ALSO

Trusted Solaris 7
Reference Manual

`label_encodings(4)`
Trusted Solaris Developer's Guide
Trusted Solaris administrator's document set
Trusted Solaris Label Administration
Trusted Solaris User's Guide

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	labelbuilder, tsol_lbuild_create, tsol_lbuild_get, tsol_lbuild_set, tsol_lbuild_destroy – Create a Motif-based user interface for interactively building a valid label or clearance
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...] #include <Dt/ModLabel.h> ModLabelData *tsol_lbuild_create(Widget widget void (*event_handler) () ok_callback lbuild_attributes extended_operation, , NULL); void *tsol_lbuild_get(ModLabelData * data, lbuild_attributes extended_operation); void tsol_lbuild_set(ModLabelData * data lbuild_attributes extended_operation, , NULL); void tsol_lbuild_destroy(ModLabelData * data);</pre>
DESCRIPTION	<p>The Label builder user interface prompts the end user for information and generates a valid CMW label, information label, sensitivity label, or clearance from the user input based on specifications in the <code>label_encodings(4)</code> file on the system where the application runs. The end user can build the label or clearance by typing a text value or by interactively choosing options.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, Label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>If the end user does not have the authorization to upgrade or downgrade labels, or if the user-built label is out of the user's accreditation range, the OK and Reset pushbuttons are grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> creates the graphical user interface and returns a pointer variable of type <code>ModLabeldata*</code> that contains information on the user interface. This information is a combination of values passed in the <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <p><code>lbuild_dialog</code> The Label builder dialog box.</p>

ok	The OK pushbutton.
cancel	The Cancel pushbutton.
reset	The Reset pushbutton.
help	The Help pushbutton.

The `tsol_lbuild_create()` parameter list takes the following values:

widget	The widget from which the dialog box is created. Any Motif widget can be passed.
ok_callback	A callback function that implements the behavior of the OK pushbutton on the dialog box.
..., NULL	A NULL terminated list of extended operations and value pairs that define the characteristics and behavior of the Label builder dialog box.

`tsol_lbuild_destroy()` destroys the `ModLabelData` structure returned by `tsol_lbuild_create()`.

`tsol_lbuild_get()` and `tsol_lbuild_set()` access the information stored in the `ModLabelData` structure returned by `tsol_lbuild_create()`.

The following extended operations can be passed to `tsol_lbuild_create()` to build the user interface, to `tsol_lbuild_get()` to retrieve information on the user interface, and to `tsol_lbuild_set()` to change the user interface information. All extended operations are valid for `tsol_lbuild_get()`, but the `*WORK*` operations are not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` because these values are set from input supplied by the end user. These exceptions are noted in the descriptions.

LBUILD_MODE

Create a user interface to build an information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

<code>LBUILD_MODE_SL</code>	Build a sensitivity label.
<code>LBUILD_MODE_CMW</code>	Build a CMW label.
<code>LBUILD_MODE_CLR</code>	Build a clearance.

LBUILD_VALUE_SL

The starting sensitivity label. This value is `ADMIN_LOW` by default and is used when the mode is `LBUILD_MODE_SL`.

LBUILD_VALUE_IL

The starting information label. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_IL .

LBUILD_VALUE_CMW

The starting CMW label. This value is ADMIN_LOW[ADMIN_LOW] by default and is used when the mode is LBUILD_MODE_CMW .

LBUILD_VALUE_CLR

The starting clearance. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_CLR .

LBUILD_USERFIELD

A character string prompt that displays at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_SHOW

Show or hide the Label builder dialog box. Value is FALSE by default.

TRUE Show the Label builder dialog box.

FALSE Hide the Label builder dialog box.

LBUILD_TITLE

A character string title that appears at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_WORK_SL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The sensitivity label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_IL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The information label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CMW

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The CMW label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CLR

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The clearance the end user is building. Value is updated to the end user's input

when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_X

The X position in pixels of the top-left corner of the Label buider dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_Y

The Y position in pixels of the top-left corner of the Label builder dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_LOWER_BOUND

The lowest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. This value is the user's minimum label.

LBUILD_UPPER_BOUND

The highest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. A supplied value should be within the user's accreditation range. If no value is specified, the value is the user's workspace sensitivity label, or if the executable has the `PRIV_SYS_TRANS_LABEL` privilege, the value is the user's clearance.

LBUILD_CHECK_AR

Check that the user-built label entered in the Update With field is within the user's accreditation range. A value of 1 means check, and a value of 0 means do not check. If checking is on and the label is out of range, an error message is raised to the end user.

LBUILD_VIEW

Use the internal or external label representation. Value is `LBUILD_VIEW_EXTERNAL` by default.

LBUILD_VIEW_INTERNAL Use the internal names for the highest and lowest labels in the system: `ADMIN_HIGH` and `ADMIN_LOW` .

LBUILD_VIEW_EXTERNAL Promote an `ADMIN_LOW` label to the next highest label, and demote an `ADMIN_HIGH` label to the next lowest label.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The `tsol_lbuild_get()` returns -1 if it is unable to get the value.

The `tsol_lbuild_create()` routine returns a variable of type `ModLabelData` that contains the information provided in the `tsol_lbuild_create()` input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface.

EXAMPLES**EXAMPLE 1** To create a Label Builder

```
(ModLabelData *)lbldata = tsol_lbuild_create(widget0, callback_function,
    LBUILD_MODE, LBUILD_MODE_CMW,
    LBUILD_TITLE, "Setting CMW Label",
    LBUILD_VIEW, LBUILD_VIEW_INTERNAL,
    LBUILD_X, 200,
    LBUILD_Y, 200,
    LBUILD_USERFIELD, "Pathname:",
    LBUILD_SHOW, FALSE,
    NULL);
```

EXAMPLE 2 To query the mode and display the Label Builder

These examples call the `tsol_lbuild_get()` routine to query the mode being used, and call the `tsol_lbuild_set()` routine so the Label builder dialog box displays.

```
mode = (int)tsol_lbuild_get(lbldata, LBUILD_MODE );

tsol_lbuild_set(lbldata, LBUILD_SHOW, TRUE,
    NULL);
```

EXAMPLE 3 To destroy the `ModLabelData` variable

This example destroys the `ModLabelData` variable returned in the call to `tsol_lbuild_create()`.

```
tsol_lbuild_destroy(lbldata);
```

FILES

```
</usr/dt/include/Dt/ModLabel.h>
```

Header file for label builder 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`label_encodings(4)`

Trusted Solaris Developer's Guide *Trusted Solaris administrator's document set*
Trusted Solaris Label Administration *Trusted Solaris User's Guide*

**SunOS 5.7 Reference
Manual****NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	labelbuilder, tsol_lbuild_create, tsol_lbuild_get, tsol_lbuild_set, tsol_lbuild_destroy – Create a Motif-based user interface for interactively building a valid label or clearance
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...] #include <Dt/ModLabel.h> ModLabelData *tsol_lbuild_create(Widget widget void (*event_handler) () ok_callback lbuild_attributes extended_operation, , NULL); void *tsol_lbuild_get(ModLabelData * data, lbuild_attributes extended_operation); void tsol_lbuild_set(ModLabelData * data lbuild_attributes extended_operation, , NULL); void tsol_lbuild_destroy(ModLabelData * data);</pre>
DESCRIPTION	<p>The Label builder user interface prompts the end user for information and generates a valid CMW label, information label, sensitivity label, or clearance from the user input based on specifications in the <code>label_encodings(4)</code> file on the system where the application runs. The end user can build the label or clearance by typing a text value or by interactively choosing options.</p> <p>Application-specific functionality is implemented in the callback for the OK pushbutton. This callback is passed to the <code>tsol_lbuild_create()</code> call where it is mapped to the OK pushbutton widget.</p> <p>When choosing options, Label builder shows the user only those classifications (and related compartments and markings) dominated by the workspace sensitivity label unless the executable has the <code>PRIV_SYS_TRANS_LABEL</code> privilege in its effective set.</p> <p>If the end user does not have the authorization to upgrade or downgrade labels, or if the user-built label is out of the user's accreditation range, the OK and Reset pushbuttons are grayed. There are no privileges to override these restrictions.</p> <p><code>tsol_lbuild_create()</code> creates the graphical user interface and returns a pointer variable of type <code>ModLabeldata*</code> that contains information on the user interface. This information is a combination of values passed in the <code>tsol_lbuild_create()</code> input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface. All information except the widget information should be accessed with the <code>tsol_lbuild_get()</code> and <code>tsol_lbuild_set()</code> routines.</p> <p>The widget information is accessed directly by referencing the following fields of the <code>ModLabelData</code> structure.</p> <p><code>lbuild_dialog</code> The Label builder dialog box.</p>

ok	The OK pushbutton.
cancel	The Cancel pushbutton.
reset	The Reset pushbutton.
help	The Help pushbutton.

The `tsol_lbuild_create()` parameter list takes the following values:

widget	The widget from which the dialog box is created. Any Motif widget can be passed.
ok_callback	A callback function that implements the behavior of the OK pushbutton on the dialog box.
..., NULL	A NULL terminated list of extended operations and value pairs that define the characteristics and behavior of the Label builder dialog box.

`tsol_lbuild_destroy()` destroys the `ModLabelData` structure returned by `tsol_lbuild_create()`.

`tsol_lbuild_get()` and `tsol_lbuild_set()` access the information stored in the `ModLabelData` structure returned by `tsol_lbuild_create()`.

The following extended operations can be passed to `tsol_lbuild_create()` to build the user interface, to `tsol_lbuild_get()` to retrieve information on the user interface, and to `tsol_lbuild_set()` to change the user interface information. All extended operations are valid for `tsol_lbuild_get()`, but the `*WORK*` operations are not valid for `tsol_lbuild_set()` or `tsol_lbuild_create()` because these values are set from input supplied by the end user. These exceptions are noted in the descriptions.

LBUILD_MODE

Create a user interface to build an information label, sensitivity label, CMW label, or clearance. Value is `LBUILD_MODE_CMW` by default.

<code>LBUILD_MODE_SL</code>	Build a sensitivity label.
<code>LBUILD_MODE_CMW</code>	Build a CMW label.
<code>LBUILD_MODE_CLR</code>	Build a clearance.

LBUILD_VALUE_SL

The starting sensitivity label. This value is `ADMIN_LOW` by default and is used when the mode is `LBUILD_MODE_SL`.

LBUILD_VALUE_IL

The starting information label. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_IL .

LBUILD_VALUE_CMW

The starting CMW label. This value is ADMIN_LOW[ADMIN_LOW] by default and is used when the mode is LBUILD_MODE_CMW .

LBUILD_VALUE_CLR

The starting clearance. This value is ADMIN_LOW by default and is used when the mode is LBUILD_MODE_CLR .

LBUILD_USERFIELD

A character string prompt that displays at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_SHOW

Show or hide the Label builder dialog box. Value is FALSE by default.

TRUE Show the Label builder dialog box.

FALSE Hide the Label builder dialog box.

LBUILD_TITLE

A character string title that appears at the top of the Label builder dialog box. Value is NULL by default.

LBUILD_WORK_SL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The sensitivity label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_IL

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The information label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CMW

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The CMW label the end user is building. Value is updated to the end user's input when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_WORK_CLR

Not valid for tsol_lbuild_set() or tsol_lbuild_create() . The clearance the end user is building. Value is updated to the end user's input

when the end user selects the Update pushbutton or interactively chooses an option.

LBUILD_X

The X position in pixels of the top-left corner of the Label buider dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_Y

The Y position in pixels of the top-left corner of the Label builder dialog box in relation to the top-left corner of the screen. By default the Label builder dialog box is positioned in the middle of the screen.

LBUILD_LOWER_BOUND

The lowest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. This value is the user's minimum label.

LBUILD_UPPER_BOUND

The highest classification (and related compartments and markings) available to the user as radio buttons for interactively building a label or clearance. A supplied value should be within the user's accreditation range. If no value is specified, the value is the user's workspace sensitivity label, or if the executable has the `PRIV_SYS_TRANS_LABEL` privilege, the value is the user's clearance.

LBUILD_CHECK_AR

Check that the user-built label entered in the Update With field is within the user's accreditation range. A value of 1 means check, and a value of 0 means do not check. If checking is on and the label is out of range, an error message is raised to the end user.

LBUILD_VIEW

Use the internal or external label representation. Value is `LBUILD_VIEW_EXTERNAL` by default.

LBUILD_VIEW_INTERNAL Use the internal names for the highest and lowest labels in the system: `ADMIN_HIGH` and `ADMIN_LOW` .

LBUILD_VIEW_EXTERNAL Promote an `ADMIN_LOW` label to the next highest label, and demote an `ADMIN_HIGH` label to the next lowest label.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

The `tsol_lbuild_get()` returns -1 if it is unable to get the value.

The `tsol_lbuild_create()` routine returns a variable of type `ModLabelData` that contains the information provided in the `tsol_lbuild_create()` input parameter list, default values for information not provided, and information on the widgets used by Label builder to create the user interface.

EXAMPLES**EXAMPLE 1** To create a Label Builder

```
(ModLabelData *)lbldata = tsol_lbuild_create(widget0, callback_function,
    LBUILD_MODE, LBUILD_MODE_CMW,
    LBUILD_TITLE, "Setting CMW Label",
    LBUILD_VIEW, LBUILD_VIEW_INTERNAL,
    LBUILD_X, 200,
    LBUILD_Y, 200,
    LBUILD_USERFIELD, "Pathname:",
    LBUILD_SHOW, FALSE,
    NULL);
```

EXAMPLE 2 To query the mode and display the Label Builder

These examples call the `tsol_lbuild_get()` routine to query the mode being used, and call the `tsol_lbuild_set()` routine so the Label builder dialog box displays.

```
mode = (int)tsol_lbuild_get(lbldata, LBUILD_MODE );

tsol_lbuild_set(lbldata, LBUILD_SHOW, TRUE,
    NULL);
```

EXAMPLE 3 To destroy the `ModLabelData` variable

This example destroys the `ModLabelData` variable returned in the call to `tsol_lbuild_create()`.

```
tsol_lbuild_destroy(lbldata);
```

FILES

```
</usr/dt/include/Dt/ModLabel.h>
```

Header file for label builder 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.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

`label_encodings(4)`

Trusted Solaris Developer's Guide *Trusted Solaris administrator's document set*
Trusted Solaris Label Administration *Trusted Solaris User's Guide*

**SunOS 5.7 Reference
Manual
NOTES**

attributes(5)

Information labels (IL s) are not supported in Trusted Solaris 7 and later releases. Trusted Solaris software interprets any IL s on communications and files from systems running earlier releases as ADMIN_LOW .

Objects still have CMW labels, and CMW labels still include the IL component: IL[SL] ; however, the IL component is fixed at ADMIN_LOW .

As a result, Trusted Solaris 7 has the following characteristics:

- IL s do not display in window labels; SL s (Sensitivity Labels) display alone within brackets.
- IL s do not float.
- Setting an IL on an object has no effect.
- Getting an object's IL will always return ADMIN_LOW .
- Although certain utilities, library functions, and system calls can manipulate IL strings, the resulting IL s are always ADMIN_LOW , and cannot be set on any objects.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmp *utmp, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfile, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

	information in any member of <code>utmpx</code> does not fit in the corresponding <code>utmp</code> member, the data is truncated. (See <code>getutent(3C)</code> for <code>utmp</code> structure)								
<code>getutmpx()</code>	The <code>getutmpx()</code> function copies the information stored in the members of the <code>utmp</code> structure to the corresponding members of the <code>utmpx</code> structure. (See <code>getutent(3C)</code> for <code>utmp</code> structure)								
<code>updwtmp()</code>	The <code>updwtmp()</code> function checks the existence of <code>wfile</code> and its parallel file, whose name is obtained by appending an "f3x" to <code>wfile</code> . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. <code>utmp</code> is written to <code>wfile</code> and the corresponding <code>utmpx</code> structure is written to the parallel file.								
<code>updwtmpx()</code>	The <code>updwtmpx()</code> function checks the existence of <code>wfilex</code> and its parallel file, whose name is obtained by truncating the final "f3x" from <code>wfilex</code> . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. <code>utmpx</code> is written to <code>wfilex</code> , and the corresponding <code>utmp</code> structure is written to the parallel file.								
RETURN VALUES	A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.								
USAGE	<p>These functions use buffered standard I/O for input, but <code>pututxline()</code> uses an unbuffered write to avoid race conditions between processes trying to modify the <code>utmpx</code> and <code>wtmpx</code> files.</p> <p>Applications should not access the <code>utmp</code> or <code>utmpx</code> database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the <code>utmp</code> and <code>utmpx</code> databases are maintained consistently.</p>								
FILES	<table> <tr> <td><code>/var/adm/utmp</code></td><td>User access and accounting information (old format)</td></tr> <tr> <td><code>/var/adm/utmpx</code></td><td>User access and accounting information (new format)</td></tr> <tr> <td><code>/var/adm/wtmp</code></td><td>History of user access and accounting information (old format)</td></tr> <tr> <td><code>/var/adm/wtmpx</code></td><td>History of user access and accounting information (new format)</td></tr> </table>	<code>/var/adm/utmp</code>	User access and accounting information (old format)	<code>/var/adm/utmpx</code>	User access and accounting information (new format)	<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)	<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)
<code>/var/adm/utmp</code>	User access and accounting information (old format)								
<code>/var/adm/utmpx</code>	User access and accounting information (new format)								
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)								
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)								
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Unsafe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Unsafe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	Unsafe								

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmp *utmp, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx() The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp() The `updwtmp()` function checks the existence of `wfile` and its parallel file, whose name is obtained by appending an “f3x” to `wfile`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmp` is written to `wfile` and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx() The `updwtmpx()` function checks the existence of `wfilex` and its parallel file, whose name is obtained by truncating the final “f3x” from `wfilex`. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. `utmpx` is written to `wfilex`, and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES	<code>/var/adm/utmp</code>	User access and accounting information (old format)
	<code>/var/adm/utmpx</code>	User access and accounting information (new format)
	<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
	<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	getutent, getutid, getutline, pututline, setutent, endutent, utmpname – Access utmp file entry
SYNOPSIS	<pre>#include <utmp.h> struct utmp *getutent(void); struct utmp *getutid(const struct utmp *id); struct utmp *getutline(const struct utmp *line); struct utmp *pututline(const struct utmp *utmp); void setutent(void); void endutent(void); int utmpname(const char *file);</pre>
DESCRIPTION	<p>The <code>getutent()</code>, <code>getutid()</code>, <code>getutline()</code>, and <code>pututline()</code> functions each return a pointer to a utmp structure with the following members:</p> <pre>char ut_user[8]; /* user login name */ char ut_id[4]; /* /sbin/inittab id (usually line #) */ char ut_line[12]; /* device name (console, lnxx) */ short ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ time_t ut_time; /* time entry was made */</pre> <p>The structure <code>exit_status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutent() The <code>getutent()</code> function reads in the next entry from a utmp-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutid() The <code>getutid()</code> function searches forward from the current point in the utmp file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code> if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the type specified in <code>id</code> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <code>id</code> \Rightarrow <code>ut_id</code>. If the end of file is reached without a match, it fails.</p> <p>getutline() The <code>getutline()</code> function searches forward from the current point in the utmp file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>ut_line</code> string matching the <code>line</code> \Rightarrow <code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>

pututline() The `pututline()` function writes the supplied `utmp` structure into the `utmp` file. It uses `getutid()` to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of `pututline()` will have searched for the proper entry using one of the these functions. If so, `pututline()` will not search. If `pututline()` does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the `utmp` structure.

When called by a process that does not have an effective uid of 0 and a sensitivity label of `ADMIN_LOW`, `pututline()` invokes a program (that has the appropriate forced privileges) to verify and write the entry, since `/etc/utmpx` is normally writable only by a process with a UID of 0 and a sensitivity label of `ADMIN_LOW`. In this event, the `ut_name` member must correspond to the actual user name associated with the process; the `ut_type` member must be either `USER_PROCESS` or `DEAD_PROCESS`; and the `ut_line` member must be a device special file and be writable by the user. If the process does not have the `PAF_TRUSTED_PATH` process attribute, all other fields in the entry are cleared.

setutent() The `setutent()` function resets the input stream to the beginning of the file. This reset should be done before each search for a new entry if it is desired that the entire file be examined.

endutent() The `endutent()` function closes the currently open file.

utmpname() The `utmpname()` function allows the user to change the name of the file examined, from `/var/adm/utmp` to any other file. It is most often expected that this other file will be `/var/adm/wtmp`. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The `utmpname()` function does not open the file but closes the old file if it is currently open and saves the new file name.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write. If the file name given is longer than 79 characters, `utmpname()` returns 0. Otherwise, it returns 1.

USAGE

These functions use buffered standard I/O for input, but `pututline()` uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the `utmp` and `wtmp` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on the `getutxent(3C)` manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
----------------------------	---

<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

`pututline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututline()` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute.

SEE ALSO

**SunOS 5.7 Reference
Manual**

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutid()` or `getutline()`, the function examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutline()` to search for multiple occurrences, it would be necessary to zero out the static area after each success, or `getutline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutent()`, `getutid()` or `getutline()` functions, if the user has just modified those contents and passed the pointer back to `pututline()`.

NAME	getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – Access utmpx file entry
SYNOPSIS	<pre>#include <utmpx.h> struct utmpx *getutxent(void); struct utmpx *getutxid(const struct utmpx *id); struct utmpx *getutxline(const struct utmpx *line); struct utmpx *pututxline(const struct utmpx *utmpx); void setutxent(void); void endutxent(void); int utmpxname(const char *file); void getutmp(struct utmpx *utmpx, struct utmp *utmp); void getutmpx(struct utmp *utmp, struct utmpx *utmpx); void updwtmp(char *wfile, struct utmp *utmp); void updwtmpx(char *wfilex, struct utmpx *utmpx);</pre>
DESCRIPTION	<p>The <code>getutxent()</code>, <code>getutxid()</code>, and <code>getutxline()</code> functions each return a pointer to a <code>utmpx</code> structure with the following members:</p> <pre>char ut_user[32]; /* user login name */ char ut_id[4]; /* /etc/inittab id (usually line #) */ char ut_line[32]; /* device name (console, lnxx) */ pid_t ut_pid; /* process id */ short ut_type; /* type of entry */ struct exit_status ut_exit; /* exit status of a process */ /* marked as DEAD_PROCESS */ struct timeval ut_tv; /* time entry was made */ long ut_session; /* session ID, used for windowing */ long pad[5]; /* reserved for future use */ short ut_syslen; /* significant length of ut_host */ /* including terminating null */ char ut_host[257]; /* host name, if remote */</pre> <p>The structure <code>exit status</code> includes the following members:</p> <pre>short e_termination; /* termination status */ short e_exit; /* exit status */</pre> <p>getutxent() The <code>getutxent()</code> function reads in the next entry from a <code>utmpx</code>-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.</p> <p>getutxid() The <code>getutxid()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry with a <code>ut_type</code> matching <code>id</code> \Rightarrow <code>ut_type</code>, if the type specified is <code>RUN_LVL</code>, <code>BOOT_TIME</code>, <code>OLD_TIME</code>, or <code>NEW_TIME</code>. If the</p>

	<p>type specified in <i>id</i> is <code>INIT_PROCESS</code>, <code>LOGIN_PROCESS</code>, <code>USER_PROCESS</code>, or <code>DEAD_PROCESS</code>, then <code>getutxid()</code> will return a pointer to the first entry whose type is one of these four and whose <code>ut_id</code> member matches <i>id</i>⇒<code>ut_id</code>. If the end of file is reached without a match, it fails.</p>
<code>getutxline()</code>	<p>The <code>getutxline()</code> function searches forward from the current point in the <code>utmpx</code> file until it finds an entry of the type <code>LOGIN_PROCESS</code> or <code>USER_PROCESS</code> which also has a <i>ut_line</i> string matching the <i>line</i>⇒<code>ut_line</code> string. If the end of file is reached without a match, it fails.</p>
<code>pututxline()</code>	<p>The <code>pututxline()</code> function writes the supplied <code>utmpx</code> structure into the <code>utmpx</code> file. It uses <code>getutxid()</code> to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of <code>pututxline()</code> will have searched for the proper entry using one of the <code>getutx()</code> routines. If so, <code>pututxline()</code> will not search. If <code>pututxline()</code> does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the <code>utmpx</code> structure.</p> <p>When called by a process that does not have an effective uid of 0 and a sensitivity label of <code>ADMIN_LOW</code>, <code>pututxline()</code> invokes a program (that has the appropriate forced privileges) to verify and write the entry, since <code>/etc/utmpx</code> is normally writable only by a process with a UID of 0 and a sensitivity label of <code>ADMIN_LOW</code>. In this event, the <code>ut_name</code> member must correspond to the actual user name associated with the process; the <code>ut_type</code> member must be either <code>USER_PROCESS</code> or <code>DEAD_PROCESS</code>; and the <code>ut_line</code> member must be a device special file and be writable by the user. If the process does not have the <code>PAF_TRUSTED_PATH</code> process attribute, all other fields in the entry are cleared.</p>
<code>setutxent()</code>	<p>The <code>setutxent()</code> function resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.</p>
<code>endutxent()</code>	<p>The <code>endutxent()</code> function closes the currently open file.</p>
<code>utmpxname()</code>	<p>The <code>utmpxname()</code> function allows the user to change the name of the file examined from <code>/var/adm/utmpx</code> to any other file, most often <code>/var/adm/wtmpx</code>. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The <code>utmpxname()</code> function does not open the file, but closes the old file if it is currently open and saves the new file name. The new file name must end with the "x" character to allow the name of the corresponding <code>utmp</code> file to be easily obtainable; otherwise, an error code of 1 is returned.</p>
<code>getutmp()</code>	<p>The <code>getutmp()</code> function copies the information stored in the members of the <code>utmpx</code> structure to the corresponding members of the <code>utmp</code> structure. If the</p>

information in any member of `utmpx` does not fit in the corresponding `utmp` member, the data is truncated. (See `getutent(3C)` for `utmp` structure)

getutmpx()

The `getutmpx()` function copies the information stored in the members of the `utmp` structure to the corresponding members of the `utmpx` structure. (See `getutent(3C)` for `utmp` structure)

updwtmp()

The `updwtmp()` function checks the existence of *wfile* and its parallel file, whose name is obtained by appending an “f3x” to *wfile* . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. *utmp* is written to *wfile* and the corresponding `utmpx` structure is written to the parallel file.

updwtmpx()

The `updwtmpx()` function checks the existence of *wfilex* and its parallel file, whose name is obtained by truncating the final “f3x” from *wfilex* . If only one of them exists, the second one is created and initialized to reflect the state of the existing file. *utmpx* is written to *wfilex* , and the corresponding `utmp` structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

USAGE

These functions use buffered standard I/O for input, but `pututxline()` uses an unbuffered write to avoid race conditions between processes trying to modify the `utmpx` and `wtmpx` files.

Applications should not access the `utmp` or `utmpx` database files directly, but should use the functions described on this manual page to interact with these files. Using these extended API s will ensure that the `utmp` and `utmpx` databases are maintained consistently.

FILES

<code>/var/adm/utmp</code>	User access and accounting information (old format)
<code>/var/adm/utmpx</code>	User access and accounting information (new format)
<code>/var/adm/wtmp</code>	History of user access and accounting information (old format)
<code>/var/adm/wtmpx</code>	History of user access and accounting information (new format)

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES****SEE ALSO****Trusted Solaris 7
Reference Manual****SunOS 5.7 Reference
Manual****NOTES**

`pututxline()` invokes a program with appropriate forced privileges to verify and write the `utmpx` structure. `pututxline` clears fields in an entry if the process does not have the `PAF_TRUSTED_PATH` process attribute

`getutent(3C)`

`ttyslot(3C)`, `utmp(4)`, `utmpx(4)`, `attributes(5)`

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either `getutxid()` or `getutxline()`, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use `getutxline()` to search for multiple occurrences it would be necessary to zero out the static after each success, or `getutxline()` would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the `getutxent()`, `getutxid()`, or `getutxline()` routines, if the user has just modified those contents and passed the pointer back to `pututxline()`.

NAME	labelclipping, Xbcltos, Xbsltos, Xbiltos, Xbcleartos – Translate a binary label and clip to the specified width
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]</pre> <pre>#include <Dt/label_clipping.h></pre> <pre>XmString xbcltos(Display *<i>display</i>, const bclabel_t *<i>cmwlabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString xbsltos(Display *<i>display</i>, const bxlabel_t *<i>senslabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString xbcleartos(Display *<i>display</i>, const bclear_t *<i>clearance</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre>
DESCRIPTION	<p>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.</p> <p><i>display</i> The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i> The CMW label to be translated.</p> <p><i>senslabel</i> The sensitivity label to be translated.</p> <p><i>clearance</i> The clearance to be translated.</p> <p><i>width</i> 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(3) man page for more information on the clipped indicator. If the specified width is equal to the display width (<i>display</i>), the label is not truncated, but word-wrapped using a width of half the display width.</p> <p><i>fontlist</i> A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i> The value of flags indicates which words in the label_encodings(4) file are used for the translation. See the bltos(3) man page for a description of the flag values: LONG_WORDS , SHORT_WORDS , LONG_CLASSIFICATION , SHORT_CLASSIFICATION , ALL_ENTRIES , ACCESS_RELATED , VIEW_EXTERNAL , VIEW_INTERNAL ,</p>

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].

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity label, information label, or clearance translated. The compound string uses the language and fonts specified in fontlist and is clipped to width . These interfaces 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.

EXAMPLES

EXAMPLE 1 To 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.

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

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.
```

SEE ALSO

Trusted Solaris 7
Reference Manual

bltos(3) , sbltos(3) , label_encodings(4)

Trusted Solaris Developer's Guide , Trusted Solaris administrator's document set , Trusted Solaris Label Administration , and Trusted Solaris User's Guide

SunOS 5.7 Reference
Manual

attributes(5) , and XmStringDraw(3X) , and FontList(3X) for information on the creation and structure of a font list.

NAME	labelclipping, Xbcltos, Xbsltos, Xbiltos, Xbcleartos – Translate a binary label and clip to the specified width
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]</pre> <pre>#include <Dt/label_clipping.h></pre> <pre>XmString xbcltos(Display *<i>display</i>, const bclabel_t *<i>cmwlabel</i>, Dimension <i>width</i>, const</pre> <pre>XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString xbsltos(Display *<i>display</i>, const bxlabel_t *<i>senslabel</i>, Dimension <i>width</i>, const</pre> <pre>XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString xbcleartos(Display *<i>display</i>, const bclear_t *<i>clearance</i>, Dimension <i>width</i>,</pre> <pre>const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre>
DESCRIPTION	<p>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.</p> <p><i>display</i> The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i> The CMW label to be translated.</p> <p><i>senslabel</i> The sensitivity label to be translated.</p> <p><i>clearance</i> The clearance to be translated.</p> <p><i>width</i> 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(3) man page for more information on the clipped indicator. If the specified width is equal to the display width (<i>display</i>), the label is not truncated, but word-wrapped using a width of half the display width.</p> <p><i>fontlist</i> A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i> The value of flags indicates which words in the label_encodings(4) file are used for the translation. See the bltos(3) man page for a description of the flag values: LONG_WORDS , SHORT_WORDS , LONG_CLASSIFICATION , SHORT_CLASSIFICATION , ALL_ENTRIES , ACCESS_RELATED , VIEW_EXTERNAL , VIEW_INTERNAL ,</p>

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].

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity label, information label, or clearance translated. The compound string uses the language and fonts specified in fontlist and is clipped to width . These interfaces 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.

EXAMPLES

EXAMPLE 1 To 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.

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

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.
```

SEE ALSO

Trusted Solaris 7
Reference Manual

bltos(3) , sbltos(3) , label_encodings(4)

Trusted Solaris Developer's Guide , Trusted Solaris administrator's document set , Trusted Solaris Label Administration , and Trusted Solaris User's Guide

SunOS 5.7 Reference
Manual

attributes(5) , and XmStringDraw(3X) , and FontList(3X) for information on the creation and structure of a font list.

NAME	labelclipping, Xbcltos, Xbsltos, Xbiltos, Xbcleartos – Translate a binary label and clip to the specified width
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]</pre> <pre>#include <Dt/label_clipping.h></pre> <pre>XmString xbcltos(Display *<i>display</i>, const bclabel_t *<i>cmwlabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString xbsltos(Display *<i>display</i>, const bxlabel_t *<i>senslabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString xbcleartos(Display *<i>display</i>, const bclear_t *<i>clearance</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre>
DESCRIPTION	<p>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.</p> <p><i>display</i> The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i> The CMW label to be translated.</p> <p><i>senslabel</i> The sensitivity label to be translated.</p> <p><i>clearance</i> The clearance to be translated.</p> <p><i>width</i> 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(3) man page for more information on the clipped indicator. If the specified width is equal to the display width (<i>display</i>), the label is not truncated, but word-wrapped using a width of half the display width.</p> <p><i>fontlist</i> A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i> The value of flags indicates which words in the label_encodings(4) file are used for the translation. See the bltos(3) man page for a description of the flag values: LONG_WORDS , SHORT_WORDS , LONG_CLASSIFICATION , SHORT_CLASSIFICATION , ALL_ENTRIES , ACCESS_RELATED , VIEW_EXTERNAL , VIEW_INTERNAL ,</p>

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].

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity label, information label, or clearance translated. The compound string uses the language and fonts specified in fontlist and is clipped to width . These interfaces 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.

EXAMPLES

EXAMPLE 1 To 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.

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

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.
```

SEE ALSO

Trusted Solaris 7
Reference Manual

bltos(3) , sbltos(3) , label_encodings(4)

Trusted Solaris Developer's Guide , Trusted Solaris administrator's document set , Trusted Solaris Label Administration , and Trusted Solaris User's Guide

SunOS 5.7 Reference
Manual

attributes(5) , and XmStringDraw(3X) , and FontList(3X) for information on the creation and structure of a font list.

NAME	labelclipping, Xbcltos, Xbsltos, Xbiltos, Xbcleartos – Translate a binary label and clip to the specified width
SYNOPSIS	<pre>cc [flag...] file... -ltsol -lDtTsol [library...]</pre> <pre>#include <Dt/label_clipping.h></pre> <pre>XmString Xbcltos(Display *<i>display</i>, const bclabel_t *<i>cmwlabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString Xbsltos(Display *<i>display</i>, const bxlabel_t *<i>senslabel</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre> <pre>XmString Xbcleartos(Display *<i>display</i>, const bclear_t *<i>clearance</i>, Dimension <i>width</i>, const XmFontList <i>fontlist</i>, const int <i>flags</i>);</pre>
DESCRIPTION	<p>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.</p> <p><i>display</i> The structure controlling the connection to an X Window System display.</p> <p><i>cmwlabel</i> The CMW label to be translated.</p> <p><i>senslabel</i> The sensitivity label to be translated.</p> <p><i>clearance</i> The clearance to be translated.</p> <p><i>width</i> 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(3) man page for more information on the clipped indicator. If the specified width is equal to the display width (<i>display</i>), the label is not truncated, but word-wrapped using a width of half the display width.</p> <p><i>fontlist</i> A list of fonts and character sets where each font is associated with a character set.</p> <p><i>flags</i> The value of flags indicates which words in the label_encodings(4) file are used for the translation. See the bltos(3) man page for a description of the flag values: LONG_WORDS , SHORT_WORDS , LONG_CLASSIFICATION , SHORT_CLASSIFICATION , ALL_ENTRIES , ACCESS_RELATED , VIEW_EXTERNAL , VIEW_INTERNAL ,</p>

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].

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWtsu
MT-Level	MT-Safe

RETURN VALUES

These interfaces return a compound string that represents the character-coded form of the CMW label, sensitivity label, information label, or clearance translated. The compound string uses the language and fonts specified in fontlist and is clipped to width . These interfaces 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.

EXAMPLES

EXAMPLE 1 To 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.

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

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.

SEE ALSO

Trusted Solaris 7
Reference Manual

bltos(3) , sbltos(3) , label_encodings(4)
Trusted Solaris Developer's Guide, *Trusted Solaris administrator's document set*, *Trusted Solaris Label Administration*, and *Trusted Solaris User's Guide*

SunOS 5.7 Reference
Manual

attributes(5) , and XmStringDraw(3X) , and FontList(3X) for information on the creation and structure of a font list.

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
DESCRIPTION	<p>These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code>], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t</code> <code>rpc_reg(const rpcprog_t prognum , const rpcvers_t versnum , const rpcproc_t procnum , char * (* procname)(), const xdrproc_t inproc , const xdrproc_t outproc , const char * nettype);</code></p> <p>Register program <i>prognum</i> , procedure <i>procname</i> , and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i> , version <i>versnum</i> , and procedure <i>procnum</i> , <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its <code>static</code> result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters while <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i> . See <code>rpc(3N)</code> . This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p> <p><code>int</code> <code>svc_reg(const SVCXPRT * xpirt , const rpcprog_t prognum , const rpcvers_t versnum , const void (* dispatch)(), const struct netconfig * netconf);</code></p> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i> . If <i>netconf</i> is <code>NULL</code> , the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i> , <i>versnum</i> , <i>netconf</i> \Rightarrow <i>nc_netid</i>] to <i>xpirt</i> \Rightarrow <i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p>

```
void svc_unreg(const rpcprog_t prognum , const rpcvers_t versnum );
```

Remove from the `rpcbind` service, all mappings of the triple [*prognum* , *versnum* , *all-transports*] to network address and all mappings within the RPC service package of the double [*prognum* , *versnum*] to dispatch routines.

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

The `PRIV_NET_SETID` privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

```
int svc_auth_reg(const int cred_flavor , const enum auth_stat (*handler)());
```

Registers the service authentication routine *handler* with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type *cred_flavor* . This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call `svc_auth_reg()` after registering the service and prior to calling `svc_run()` . When needed to process an RPC credential of type *cred_flavor* , the *handler* procedure will be called with two parameters (`struct svc_req * rqst` , `struct rpc_msg * msg`) and is expected to return a valid `enum auth_stat` value. There is no provision to change or delete an authentication handler once registered.

The `svc_auth_reg()` routine returns 0 if the registration is successful, 1 if *cred_flavor* already has an authentication handler registered for it, and -1 otherwise.

```
void xpirt_register(const SVCXPRT * xprt );
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3N)`). Service implementors usually do not need this routine.

```
void xpirt_unregister(const SVCXPRT * xprt );
```

Before an RPC service transport handle *xprt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable `svc_fdset` [see `rpc_svc_calls(3N)`]. Service implementors usually do not need this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

**SUMMARY
OF TRUSTED
SOLARIS
CHANGES**

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is on when `rpc_reg()` or `rpc_svc()` is called, a multilevel mapping is created. To delete a multilevel mapping, `svc_unreg()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The `PRIV_NET_SETID` privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

**Trusted Solaris 7
Reference Manual**

**SunOS 5.7 Reference
Manual**

`inetd(1M)`, `rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`,
`rpc_svc_create(3N)`, `rpcbind(3N)`

`select(3C)`, `rpc_svc_err(3N)`, `attributes(5)`

NAME	rpc_svc_reg, rpc_reg, svc_reg, svc_unreg, svc_auth_reg, xpirt_register, xpirt_unregister – Library routines for registering servers
DESCRIPTION	<p>These routines are a part of the RPC library which allows the RPC servers to register themselves with <code>rpcbind()</code> [see <code>rpcbind(1M)</code>], and associate the given program and version number with the dispatch function. When the RPC server receives an RPC request, the library invokes the dispatch routine with the appropriate arguments.</p>
Routines	<p>See <code>rpc(3N)</code> for the definition of the <code>SVCXPRT</code> data structure.</p> <pre>#include rpc/rpc.h</pre> <p><code>bool_t rpc_reg(const rpcprog_t prognum, const rpcvers_t versnum, const rpcproc_t procnum, char * (* procname)(), const xdrproc_t inproc, const xdrproc_t outproc, const char * nettype);</code></p> <p>Register program <i>prognum</i>, procedure <i>procname</i>, and version <i>versnum</i> with the RPC service package. If a request arrives for program <i>prognum</i>, version <i>versnum</i>, and procedure <i>procnum</i>, <i>procname</i> is called with a pointer to its parameter(s); <i>procname</i> should return a pointer to its static result(s). The <i>arg</i> parameter to <i>procname</i> is a pointer to the (decoded) procedure argument. <i>inproc</i> is the XDR function used to decode the parameters while <i>outproc</i> is the XDR function used to encode the results. Procedures are registered on all available transports of the class <i>nettype</i>. See <code>rpc(3N)</code>. This routine returns 0 if the registration succeeded, -1 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p> <p><code>int svc_reg(const SVCXPRT * xpirt, const rpcprog_t prognum, const rpcvers_t versnum, const void (* dispatch)(), const struct netconfig * netconf);</code></p> <p>Associates <i>prognum</i> and <i>versnum</i> with the service dispatch procedure, <i>dispatch</i>. If <i>netconf</i> is <code>NULL</code>, the service is not registered with the <code>rpcbind</code> service. For example, if a service has already been registered using some other means, such as <code>inetd</code> (see <code>inetd(1M)</code>), it will not need to be registered again. If <i>netconf</i> is non-zero, then a mapping of the triple [<i>prognum</i>, <i>versnum</i>, <i>netconf</i> \Rightarrow <i>nc_netid</i>] to <i>xpirt</i> \Rightarrow <i>xp_ltaddr</i> is established with the local <code>rpcbind</code> service.</p> <p>The <code>svc_reg()</code> routine returns 1 if it succeeds, and 0 otherwise.</p> <p>If the server has the <code>PRIV_NET_MAC_READ</code> privilege, a multilevel mapping is created. If the mapping is being established to a transport that uses a privileged address, the server must have the <code>PRIV_NET_PRIVADDR</code> privilege.</p>

```
void svc_unreg(const rpcprog_t prognum , const rpcvers_t versnum );
```

Remove from the `rpcbind` service, all mappings of the triple [*prognum* , *versnum* , *all-transport*] to network address and all mappings within the RPC service package of the double [*prognum* , *versnum*] to dispatch routines.

If the server has the `PRIV_NET_MAC_READ` privilege, a multilevel mapping is created. If the mapping being deleted is to a transport that uses a privileged address, the server must have the `PRIV_NET_PRIVADDR` privilege.

The `PRIV_NET_SETID` privilege is required in order for anyone other than the owner of a mapping to delete the mapping.

```
int svc_auth_reg(const int cred_flavor , const enum auth_stat (*handler)());
```

Registers the service authentication routine *handler* with the dispatch mechanism so that it can be invoked to authenticate RPC requests received with authentication type *cred_flavor* . This interface allows developers to add new authentication types to their RPC applications without needing to modify the libraries. Service implementors usually do not need this routine.

Typical service application would call `svc_auth_reg()` after registering the service and prior to calling `svc_run()` . When needed to process an RPC credential of type *cred_flavor* , the *handler* procedure will be called with two parameters (`struct svc_req * rqst` , `struct rpc_msg * msg`) and is expected to return a valid `enum auth_stat` value. There is no provision to change or delete an authentication handler once registered.

The `svc_auth_reg()` routine returns 0 if the registration is successful, 1 if *cred_flavor* already has an authentication handler registered for it, and -1 otherwise.

```
void xpirt_register(const SVCXPRT * xprt );
```

After RPC service transport handle *xprt* is created, it is registered with the RPC service package. This routine modifies the global variable `svc_fdset` (see `rpc_svc_calls(3N)`). Service implementors usually do not need this routine.

```
void xpirt_unregister(const SVCXPRT * xprt );
```

Before an RPC service transport handle *xprt* is destroyed, it unregisters itself with the RPC service package. This routine modifies the global variable `svc_fdset` [see `rpc_svc_calls(3N)`]. Service implementors usually do not need this routine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Most `rpcbind` services operate only on mappings that either match the sensitivity label of the server or are multilevel.

The `PRIV_NET_MAC_READ` privilege affects the operation of several `rpcbind` services. If the privilege is on when `rpc_reg()` or `rpc_svc()` is called, a multilevel mapping is created. To delete a multilevel mapping, `svc_unreg()` must be called with the privilege on.

The `PRIV_NET_PRIVADDR` privilege is required for `rpc_reg()`, `rpc_svc()`, or `svc_unreg()` calls that create or delete mappings for a transport that uses a privileged address.

The `PRIV_NET_SETID` privilege is required by `svc_unreg()` in order for anyone other than the owner of a mapping to delete the mapping.

SEE ALSO

Trusted Solaris 7
Reference Manual

SunOS 5.7 Reference
Manual

`inetd(1M)`, `rpcbind(1M)`, `rpc(3N)`, `rpc_svc_calls(3N)`,
`rpc_svc_create(3N)`, `rpcbind(3N)`

`select(3C)`, `rpc_svc_err(3N)`, `attributes(5)`

NAME	XTSOLgetClientAttributes – Get all CMW attributes associated with a client							
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLgetClientAttributes(<i>display</i> , <i>windowid</i> , <i>clientattrp</i>); Display * <i>display</i> ; XID <i>windowid</i> ; XTSOLClientAttributes * <i>clientattrp</i> ;							
DESCRIPTION	XTSOLgetClientAttributes() is used to get all CMW attributes associated with a client in a single call. The attributes include process ID, user ID, IP address, audit flags and session ID.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>windowid</i>	Specifies window ID of X client.						
	<i>clientattrp</i>	Client must provide a pointer to an XTSOLClientAttributes structure.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadValue	Not a valid client						
SEE ALSO	XTSOLgetPropAttributes(3X11TSOL), XTSOLgetResAttributes(3X11TSOL)							

NAME	XTSOLgetPropAttributes – Get all CMW attributes associated with a property hanging on a window							
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLgetPropAttributes(<i>display</i> , <i>window</i> , <i>property</i> , <i>cmwpropattrp</i>); Display * <i>display</i> ; Window <i>window</i> ; Atom <i>property</i> ; XTSOLPropAttributes * <i>cmwpropattrp</i> ;							
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetPropAttributes() is used to get all CMW attributes associated with a property hanging out of a window in a single call. The attributes include UID, information label, and sensitivity label.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>window</i>	Specifies the ID of a window system object.						
	<i>property</i>	Specifies the property atom.						
	<i>cmwwinattrp</i>	Client must provide a pointer to XTSOLPropAttributes.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
SEE ALSO	XTSOLgetClientAttributes(3X11TSOL), XTSOLgetResAttributes(3X11TSOL)							

NAME	XTSOLgetPropLabel – Get the CMW label associated with a property hanging on a window							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLgetPropLabel(<i>display</i>, <i>window</i>, <i>property</i>, <i>cmwlabel</i>); Display *<i>display</i>; Window <i>window</i>; Atom <i>property</i>; bclabel_t *<i>cmwlabel</i>;</pre>							
DESCRIPTION	Client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetPropLabel() is used to get the CMW label associated with a property hanging on a window.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>window</i>	Specifies the ID of the window whose property’s CMW label you want to get.						
	<i>property</i>	Specifies the property atom.						
	<i>cmwlabel</i>	Returns a CMW label that is the current CMW label of the specified property. This label contains an SL. Client needs to provide a bclabel_t type storage and passes the address of this storage as the function argument. Client must provide a pointer to bclabel_t.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
SEE ALSO	XTSOLgetPropAttributes(3X11TSOL), XTSOLsetPropLabel(3X11TSOL)							

NAME	XTSOLgetPropUID – Get the UID associated with a property hanging on a window							
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLgetPropUID(<i>display</i> , <i>window</i> , <i>property</i> , <i>uidp</i>); Display * <i>display</i> ; Window <i>window</i> ; Atom <i>property</i> ; uid_t * <i>uidp</i> ;							
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetPropUID() gets the ownership of a window’s property. This allows a client to get the ownership of an object it did not create.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>window</i>	Specifies the ID of the window whose property’s UID you want to get.						
	<i>property</i>	Specifies the property atom.						
	<i>uidp</i>	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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
SEE ALSO	XTSOLgetPropAttributes(3X11TSOL), XTSOLsetPropUID(3X11TSOL)							

NAME	XTSOLgetResAttributes – Get all CMW attributes associated with a window or a pixmap							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> Status XTSOLgetResAttributes(<i>display</i>, <i>object</i>, <i>type</i>, <i>cmwwinattrp</i>); Display *<i>display</i>; XID <i>object</i>; ResourceType <i>type</i>; XTSOLResAttributes *<i>cmwwinattrp</i>;</pre>							
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetResAttributes() is used to get all CMW attributes associated with a window or a pixmap in a single call. The attributes include UID, information label, sensitivity label, input information label, and workstation owner.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>object</i>	Specifies the ID of a window system object. Possible window system objects are windows and pixmaps.						
	<i>type</i>	Specifies what type of resource is being accessed. Possible values are IsWindow and IsPixmap						
	<i>cmwwinattrp</i>	Client must provide a pointer to XTSOLResAttributes.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadPixmap	Not a valid pixmap						
	BadValue	Not a valid type						
SEE ALSO	XTSOLgetClientAttributes(3X11TSOL), XTSOLgetPropAttributes(3X11TSOL)							

NAME	XTSOLgetResLabel – Get the CMW label associated with a window, a pixmap, or a colormap							
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLgetResLabel(<i>display, object, type, cmwlabel</i>); Display * <i>display</i> ; XID <i>object</i> ; ResourceType <i>type</i> ; bclabel_t * <i>cmwlabel</i> ;							
DESCRIPTION	The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges. XTSOLgetResLabel() is used to get the CMW label associated with a window or a pixmap or a colormap.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>object</i>	Specifies the ID of a window system object whose CMW label you want to get. Possible window system objects are windows, and pixmaps or colormaps.						
	<i>type</i>	Specifies what type of resource is being accessed. Possible values are IsWindow, IsPixmap or IsColormap.						
	<i>cmwlabel</i>	Returns a CMW label which is the current CMW label of the specified object. This label contains an SL. Client needs to provide a bclabel_t type storage and passes the address of this storage as the function argument. Client must provide a pointer to bclabel_t.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadPixmap	Not a valid pixmap						
	BadValue	Not a valid type						
SEE ALSO	XTSOLgetClientAttributes(3X11TSOL), XTSOLsetResLabel(3X11TSOL)							

NAME	XTSOLgetResUID – Get the UID associated with a window, a pixmap							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> Status XTSOLgetResUID(<i>display</i>, <i>object</i>, <i>type</i>, <i>uidp</i>); Display *<i>display</i>; XID <i>object</i>; ResourceType <i>type</i>; uid_t *<i>uidp</i>;</pre>							
DESCRIPTION	<p>The client requires the PRIV_WIN_DAC_READ and PRIV_WIN_MAC_READ privileges.</p> <p>XTSOLgetResUID() gets the ownership of a window system object. This allows a client to get the ownership of an object it did not create.</p>							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>object</i>	Specifies the ID of a window system object whose UID you want to get. Possible window system objects are windows or pixmaps.						
	<i>type</i>	Specifies what type of resource is being accessed. Possible values are IsWindow and IsPixmap.						
	<i>uidp</i>	Returns a UID which is the current UID of the specified object. Client must provide a pointer to uid_t.						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadPixmap	Not a valid pixmap						
	BadValue	Not a valid type						
SEE ALSO	XTSOLgetClientAttributes(3X11TSOL), XTSOLgetResAttributes(3X11TSOL), XTSOLgetResUID(3X11TSOL)							

NAME	XTSOLgetSSHeight – Get the height of screen stripe							
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLgetSSHeight(<i>display</i> , <i>screen_num</i> , <i>newheight</i>); Display * <i>display</i> ; int <i>screen_num</i> ; int * <i>newheight</i> ;							
DESCRIPTION	XTSOLgetSSHeight() 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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>screen_num</i>	Specifies the screen number.						
	<i>newheight</i>	Specifies the storage area where the height of the stripe in pixels is returned.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadValue	Not a valid <i>screen_num</i> or <i>newheight</i>						
SEE ALSO	XTSOLsetSSHeight(3X11TSOL)							

NAME	XTSOLgetWorkstationOwner – Get the ownership of the workstation							
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLgetWorkstationOwner (<i>display</i> , <i>uidp</i>); Display * <i>display</i> ; uid_t * <i>uidp</i> ;							
DESCRIPTION	XTSOLgetWorkstationOwner() is used to get the ownership of the workstation.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>uidp</i>	Returns a UID which is the current UID of the specified Display workstation server. Client must provide a pointer to uid_t.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None.							
ERRORS	BadAccess	Lack of privilege						
SEE ALSO	XTSOLsetWorkstationOwner(3X11TSOL)							

NAME	XTSOLIsWindowTrusted – Test if a window is created by a trusted client							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLIsWindowTrusted(display, window); Display *display; Window *window;</pre>							
DESCRIPTION	XTSOLIsWindowTrusted() 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	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>window</i>	Specifies the ID of the window to be tested.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	True	if the window is created by a trusted client.						
ERRORS	BadWindow	Not a valid window						

NAME	XTSOLMakeTPWindow – Make this window a Trusted Path window								
SYNOPSIS	#include <tsol/Xtsol.h> Status XTSOLMakeTPWindow (<i>display</i> , <i>w</i>); Display * <i>display</i> ; Window * <i>w</i> ;								
DESCRIPTION	XTSOLMakeTPWindow() is used to make 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.								
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().							
	<i>w</i>	Specifies the ID of a window.							
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>			ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWxwplt (available only on Trusted Solaris systems)								
MT-Level	MT-Unsafe								
RETURN VALUES	None								
ERRORS	BadAccess	Lack of privilege							
	BadWindow	Not a valid window							
	BadValue	Not a valid type							

NAME	XTSOLsetPolyInstInfo – Set polyinstantiation information								
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLsetPolyInstInfo(display, sl, uidp, enabled); Display *display; bslabel_t sl; uid_t *uidp; int enabled;								
DESCRIPTION	XTSOLsetPolyInstInfo() 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 <i>sl</i> and <i>uid</i> a client can use this call to set the SL and UID with <i>enabled</i> flag to TRUE. The client should also restore the <i>enabled</i> flag to FALSE after retrieving the property value. Client must have the PRIV_WIN_MAC_WRITE and PRIV_WIN_DAC_WRITE privileges.								
PARAMETERS	display	Specifies 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.							
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:								
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>			ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
Availability	SUNWxwplt (available only on Trusted Solaris systems)								
MT-Level	MT-Unsafe								
RETURN VALUES	None								
ERRORS	BadAccess	Lack of privilege							
	BadValue	Not a valid <i>display</i> or <i>sl</i> .							

NAME	XTSOLsetPropLabel – Set the CMW label associated with a property hanging on a window							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetPropLabel(<i>*display</i>, <i>window</i>, <i>property</i>, <i>*cmwlabel</i>, <i>labelFlag</i>); Display <i>*display</i>; Window <i>window</i>; Atom <i>property</i>; bclabel_t <i>*cmwlabel</i>; enum setting_flag <i>flag</i>;</pre>							
DESCRIPTION	XTSOLsetPropLabel() is used to change the CMW label 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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>window</i>	Specifies the ID of the window whose property’s CMW label you want to change.						
	<i>property</i>	Specifies the property atom.						
	<i>cmwlabel</i>	Specifies a pointer to a CMW label structure which contains a CMW label. Only a portion (depends on <i>labelFlag</i>) of the CMW label needs to be specified. The unspecified portion of the CMW label is not interpreted by the server.						
	<i>labelFlag</i>	Specifies which portion of the CMW label will be changed. Possible values are: SETCL_ALL and SETCL_SL.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
	BadValue	Not a valid <i>labelFlag</i> or <i>cmwlabel</i>						

SEE ALSO

XTSOLgetPropAttributes(3X11TSOL), XTSOLgetPropLabel(3X11TSOL)

NAME	XTSOLsetPropUID – Set the UID associated with a property hanging on a window							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetPropUID(<i>display</i>, <i>window</i>, <i>property</i>, <i>uidp</i>); Display *<i>display</i>; Window <i>window</i>; Atom <i>property</i>; uid_t *<i>uidp</i>;</pre>							
DESCRIPTION	XTSOLsetPropUID() 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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>window</i>	Specifies the ID of the window whose property’s UID you want to change.						
	<i>property</i>	Specifies the property atom.						
	<i>uidp</i>	Specifies a pointer to a uid_t that contains a UID.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadAtom	Not a valid atom						
SEE ALSO	XTSOLgetPropAttributes(3X11TSOL), XTSOLgetPropUID(3X11TSOL)							

NAME	XTSOLsetResLabel – Set the CMW label associated with a window or a pixmap							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetResLabel(<i>display</i>, <i>object</i>, <i>type</i>, <i>cmwlabel</i>, <i>labelFlag</i>); Display *<i>display</i>; XID <i>object</i>; ResourceType <i>type</i>; bclabel_t *<i>cmwlabel</i>; enum setting_flag(<i>labelFlag</i>);</pre>							
DESCRIPTION	<p>The client must have the PRIV_WIN_DAC_WRITE, PRIV_WIN_MAC_WRITE, PRIV_WIN_UPGRADE_SL, and PRIV_WIN_DOWNGRADE_SL privileges.</p> <p>XTSOLsetResLabel() is used to change the CMW label associated with a window or a pixmap.</p>							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>object</i>	Specifies the ID of a window system object whose CMW label you want to change. Possible window system objects are windows and pixmaps. The CMW label is not interpreted by the server.						
	<i>labelFlag</i>	Specifies which portion of the CMW label will be changed. Possible values are: RES_ALL, and RES_SL.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadPixmap	Not a valid pixmap						
	BadValue	Not a valid <i>type</i> , <i>labelFlag</i> , or <i>cmwlabel</i>						
SEE ALSO	XTSOLgetResAttributes(3X11TSOL), XTSOLgetResLabel(3X11TSOL)							

NAME	XTSOLsetResUID – Set the UID associated with a window, a pixmap, or a colormap							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> Status XTSOLsetResUID(<i>display</i>, <i>object</i>, <i>type</i>, <i>uidp</i>); Display *<i>display</i>; XID <i>object</i>; ResourceType <i>type</i>; uid_t *<i>uidp</i>;</pre>							
DESCRIPTION	<p>The client must have the PRIV_WIN_DAC_WRITE and PRIV_WIN_MAC_WRITE privileges. XTSOLsetResUID() 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.</p>							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>object</i>	Specifies the ID of a window system object whose UID you want to change. Possible window system objects are windows and pixmaps.						
	<i>type</i>	Specifies what type of resource is being accessed. Possible values are: IsWindow and IsPixmap.						
	<i>uidp</i>	Specifies a pointer to a uid_t structure that contains a UID.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadWindow	Not a valid window						
	BadPixmap	Not a valid pixmap						
	BadValue	Not a valid type						
SEE ALSO	XTSOLgetResUID(3X11TSOL)							

NAME	XTSOLsetSessionHI – Set the session high sensitivity label to the window server						
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetSessionHI(<i>display</i>, <i>sl</i>); Display *<i>display</i>; bslabel_t *<i>sl</i>;</pre>						
DESCRIPTION	XTSOLsetSessionHI() After the session high label has been set by a Trusted Solaris window system TCB component, logintool, Xsun will reject connection request from clients running at higher sensitivity labels than the session high label. The client must have the PRIV_WIN_CONFIG privilege.						
PARAMETERS	<p><i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay().</p> <p><i>sl</i> Specifies a pointer to a sensitivity label to be used as the session HIGH label.</p>						
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr> <tr> <td>MT-Level</td><td>MT-Unsafe</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
Availability	SUNWxwplt (available only on Trusted Solaris systems)						
MT-Level	MT-Unsafe						
RETURN VALUES	None						
ERRORS	BadAccess Lack of privilege						
SEE ALSO	XTSOLsetSessionLO(3X11TSOL)						

NAME	XTSOLsetSessionLO - Set the session low sensitivity label to the window server							
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLsetSessionLO(<i>display</i>, <i>sl</i>); Display *<i>display</i>; bslabel_t *<i>sl</i>;</pre>							
DESCRIPTION	XTSOLsetSessionLO() sets the session low sensitivity label. After the session low label has been set by a Trusted Solaris window system TCB component, logintool, Xsun 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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay().						
	<i>sl</i>	Specifies a pointer to a sensitivity label to be used as the session low label.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
SEE ALSO	XTSOLsetSessionHI(3X11TSOL)							

NAME	XTSOLsetSSHeight – Set the height of screen stripe							
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLsetSSHeight(<i>display</i> , <i>screen_num</i> , <i>newheight</i>); Display * <i>display</i> ; int <i>screen_num</i> ; int <i>newheight</i> ;							
DESCRIPTION	XTSOLsetSSHeight() 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.							
PARAMETERS	<i>display</i>	Specifies a pointer to the Display structure; returned from XOpenDisplay.						
	<i>screen_num</i>	Specifies the screen number.						
	<i>newheight</i>	Specifies the height of the stripe in pixels.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
	BadValue	Not a valid <i>screen_num</i> or <i>newheight</i> .						
SEE ALSO	XTSOLgetSSHeight(3X11TSOL)							

NAME	XTSOLsetWorkstationOwner – Set the ownership of the workstation							
SYNOPSIS	#include <tsol/Xtsol.h> XTSOLsetWorkstationOwner(display, uidp); Display *display; uid_t *uidp; XTSOLClientAttributes *clientattrp;							
DESCRIPTION	XTSOLsetWorkstationOwner() is used by Trusted Solaris 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.							
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.						
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:							
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>Availability</td><td>SUNWxwplt (available only on Trusted Solaris systems)</td></tr><tr><td>MT-Level</td><td>MT-Unsafe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWxwplt (available only on Trusted Solaris systems)	MT-Level	MT-Unsafe
ATTRIBUTE TYPE	ATTRIBUTE VALUE							
Availability	SUNWxwplt (available only on Trusted Solaris systems)							
MT-Level	MT-Unsafe							
RETURN VALUES	None							
ERRORS	BadAccess	Lack of privilege						
SEE ALSO	XTSOLgetWorkstationOwner(3X11TSOL)							

NAME	XTSOLShutdown – Shut down the system
SYNOPSIS	<pre>#include <tsol/Xtsol.h> XTSOLShutdown(<i>display</i>); Display * <i>display</i></pre>
DESCRIPTION	XTSOLShutdown() shuts down the system of <i>display</i> . The client must have the PRIV_SYS_BOOT privilege to perform this operation.
PARAMETERS	<i>display</i> Specifies a pointer to the Display structure; returned from XOpenDisplay().
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWxwplt (available only on Trusted Solaris systems)
MT-Level	MT-Unsafe

Index

A

accept — accept a connection on a socket 120
accept(3N) 97
access utmpx file entry
 — endutxent 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — getutmp 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — getutmpx 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — getutxent 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — getutxid 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — getutxline 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — pututxline 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — setutxent 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — updwtmp 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — updwtmpx 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
 — utmpxname 430, 545, 549, 553, 557, 561,
 786, 979, 1218, 1222, 1229
adornfc — adorn the final component of a
 pathname 122
au_preselect — preselect an audit record 137
au_user_mask — get user's binary preselection
 mask 143
audit control file information

 — endac 409, 449, 451, 453, 455, 457, 942
 — getacdir 409, 449, 451, 453, 455, 457, 942
 — getacflg 409, 449, 451, 453, 455, 457, 942
 — getacinfo 409, 449, 451, 453, 455, 457,
 942
 — getacmin 409, 449, 451, 453, 455, 457,
 942
 — getacna 409, 449, 451, 453, 455, 457, 942
 — setac 409, 449, 451, 453, 455, 457, 942
audit record tokens, manipulating
 — au_preselect 137
auditwrite
 auditwrite 124
auditwrite error messages
 aw_errno 145–147
 aw_perror 145–147
 aw_strerror 145–147
auth_set_to_str — places the name of
 each authorization into a
 string 139, 141, 273, 434, 494,
 1008, 1010
auth_to_str — returns a pointer to the text
 authorization name 139, 141,
 273, 434, 494, 1008, 1010

B

Basic Security Module functions
 — au_preselect 137
 — au_user_mask 143
bclearhigh — initialize Admin High Binary
 Clearance 148, 150, 162, 166,

168, 188, 195, 197, 211, 231, 252, 254, 266

bclearlow — initialize Admin Low Binary Clearance 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bcleartoh — Binary Clearance to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

bcleartoh_r — Binary Clearance to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

bcleartos — Binary Clearance to string 158, 182, 207, 244, 262

bclearundef — initialize Undefined Binary Clearance 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bclearvalid — check validity of binary clearance 164, 213, 250, 268

bclhigh — initialize Admin High Binary CMW Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bcllow — initialize Admin Low Binary CMW Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bcltobanner — translate Binary CMW Label to printer banner page fields 170

bcltoh — Binary CMW Label to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

bcltoh_r — Binary CMW Label to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

bcltoil — reference Information Label 180, 186, 205, 236, 500, 502, 953, 955

bcltos — Binary CMW Label to string 158, 182, 207, 244, 262

bcltosl — reference Sensitivity Label 180, 186, 205, 236, 500, 502, 953, 955

bclundef — initialize Undefined Binary CMW Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bilconjoin — conjoin two binary information labels 190

bildominates — compare Information Labels for dominance 191, 193, 215, 217, 221, 223, 225, 227, 238

bilequal — compare Information Labels for equality 191, 193, 215, 217, 221, 223, 225, 227, 238

bilhigh — initialize Admin High Binary Information Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

billow — initialize Admin Low Binary Information Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bilttoh — Binary Information Label to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

bilttoh_r — Binary Information Label to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

biltolev — reference Information Level 180, 186, 205, 236, 500, 502, 953, 955

biltos — Binary Information Label to string 158, 182, 207, 244, 262

bilundef — initialize Undefined Binary Information Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bimdominates — compare Information Label Markings Sets for dominance 191, 193, 215, 217, 221, 223, 225, 227, 238

bimequal — compare Information Label Markings Sets for equality 191, 193, 215, 217, 221, 223, 225, 227, 238

bind — bind a name to a socket 219

bldominates — compare levels for dominance 191, 193, 215, 217, 221, 223, 225, 227, 238

blequal — compare levels for equality 191, 193, 215, 217, 221, 223, 225, 227, 238

blinrange — compare level to be between bounding levels 191, 193, 215, 217, 221, 223, 225, 227, 238
blinset — check level for set inclusion 229
blmanifest — create manifest binary labels 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266
blmaximum — least upper bound of two binary Levels 233–235
blminimum — greatest lower bound two binary Levels 233–235
blminmax — bound of two binary levels 233–235
blportion — access binary label portions 180, 186, 205, 236, 500, 502, 953, 955
blstrictdom — compare levels for strict dominance 191, 193, 215, 217, 221, 223, 225, 227, 238
bltcolor — get character-coded color name of label 240, 242
bltcolor_r — get character-coded color name of label 240, 242
bltype — check type of Binary Label 248, 951
blvalid — check validity of binary label 164, 213, 250, 268
bslhigh — initialize Admin High Binary Sensitivity Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266
bsllow — initialize Admin Low Binary Sensitivity Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266
bsltoh — Binary Sensitivity Label to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574
bsltoh_r — Binary Sensitivity Label to hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574
bsltos — Binary Sensitivity Label to string 158, 182, 207, 244, 262
bslundef — initialize Undefined Binary Sensitivity Label 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266

bslvalid — check validity of binary sensitivity label 164, 213, 250, 268
btohex — convert binary label to hexadecimal 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

C

chkauth — verifies that a user has authorization specified by auth_id 139, 141, 273, 434, 494, 1008, 1010
clnt_call — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
clnt_control — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
clnt_create — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
clnt_create_timed — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
clnt_create_vers — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
clnt_create_vers_timed — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
clnt_destroy — library routines for dealing with creation and manipulation of CLIENT handles 279, 285,

291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_dg_create` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_freeres` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892

`clnt_geterr` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892

`clnt_pcreateerror` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_perrno` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892

`clnt_perror` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892

`clnt_raw_create` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_screateerror` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_sperrno` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892

`clnt_sperror` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892

`clnt_tli_create` — library routines for dealing with creation and

manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_tp_create` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_tp_create_timed` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clnt_vc_create` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

`clock_getres` — high-resolution clock operations 387, 389, 391

`clock_gettime` — high-resolution clock operations 387, 389, 391

`clock_settime` — high-resolution clock operations 387, 389, 391

communications

- accept a connection on a socket — accept 120
- bind a name to a socket — bind 219
- listen for connections on a socket — listen 605
- send a message from a socket — send, sendto, sendmsg 936, 938, 940

create a door descriptor — `door_create` 405

current working directory

- get pathname — `mldmldgetcwd` 607

D

directories

- get pathname of current working directory — `mldgetcwd` 607

dn_comp — resolver routines 393, 399, 792, 798, 804, 810, 816, 822, 828, 834, 840

dn_expand — resolver routines 393, 399, 792, 798, 804, 810, 816, 822, 828, 834, 840

door_create — create a door descriptor 405

door_tcred — Return the extended credential information associated with the client of the current door invocation 407

E

endac — get audit control file information 409, 449, 451, 453, 455, 457, 942

endauclass — close audit_class database file 411, 459, 461, 463, 465, 944

endauevent — close audit_event database file 413, 473, 476, 479, 482, 485, 488, 491, 946

endauuser — get audit_user database entry 416, 496, 498, 949

endprofent — ends profile entry retrieval operations 418, 436, 511, 514, 963

endutent — access utmp file entry 427, 536, 539, 542, 783, 976, 1226

endutxent — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

F

file status

- get — mldstat, mldlstat 609, 615

file tree

- recursively descend — nftw 445, 625

free_auth_set — releases the memory returned by str_to_auth_set 139, 141, 273, 434, 494, 1008, 1010

free_profent — frees pointers returned by getprofentbyname or getprofent 418, 436, 511, 514, 963

ftw — walk a file tree 445, 625

Index-1273

G

get

- user audit characteristics for socket or TLI peer— getpeerinfo 506

get_auth_text — returns a pointer to the authorization description text 139, 141, 273, 434, 494, 1008, 1010

getacdir — get audit control file information 409, 449, 451, 453, 455, 457, 942

getacflg — get audit control file information 409, 449, 451, 453, 455, 457, 942

getacinfo — get audit control file information 409, 449, 451, 453, 455, 457, 942

getacmin — get audit control file information 409, 449, 451, 453, 455, 457, 942

getacna — get audit control file information 409, 449, 451, 453, 455, 457, 942

getauclassent — get audit_class database entry 411, 459, 461, 463, 465, 944

getauclassent_r — get audit_class database entry 411, 459, 461, 463, 465, 944

getauclassnam — get audit_class database entry 411, 459, 461, 463, 465, 944

getauclassnam_r — get audit_class database entry 411, 459, 461, 463, 465, 944

getauditflags() — generate process audit state 504

getauditflagsbin — convert audit flag specifications 467, 469, 471

getauditflagschar — convert audit flag specifications 467, 469, 471

getauevent — get audit_event database entry 413, 473, 476, 479, 482, 485, 488, 491, 946

getauevent_r — get audit_event database entry 413, 473, 476, 479, 482, 485, 488, 491, 946

getauevnam — get audit_event database entry 413, 473, 476, 479, 482, 485, 488, 491, 946

getauevnam_r — get audit_event database entry 413, 473, 476, 479, 482, 485, 488, 491, 946

getauevnonam — get audit_event database entry 413, 473, 476, 479, 482, 485, 488, 491, 946

getauevnum — get audit_event database entry 413, 473, 476, 479, 482, 485, 488, 491, 946

getauevnum_r — get audit_event database entry 413, 473, 476, 479, 482, 485, 488, 491, 946

getauuserent — get audit_user database entry 416, 496, 498, 949

getauuserent_r — get audit_user database entry 416, 496, 498, 949

getauusernam — get audit_user database entry 416, 496, 498, 949

getauusernam_r — get audit_user database entry 416, 496, 498, 949

getcil — extract Information Label 180, 186, 205, 236, 500, 502, 953, 955

getcsl — extract Sensitivity Label 180, 186, 205, 236, 500, 502, 953, 955

getpeerinfo 506

getprofent — enumerates profile entries from the database 418, 436, 511, 514, 963

getprofentbyname — searches for information for a profile with the specified name 418, 436, 511, 514, 963

getsockopt — get options on sockets 523, 969

getutent — access utmp file entry 427, 536, 539, 542, 783, 976, 1226

getutid — access utmp file entry 427, 536, 539, 542, 783, 976, 1226

getutline — access utmp file entry 427, 536, 539, 542, 783, 976, 1226

getutmp — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutmpx — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutxent — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

endutxent() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutmp() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutmpx() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutxent() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutxid() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutxline() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

pututxline() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

setutxent() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

updwtmp() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

updwtmpx() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

utmpxname() 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutxid — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getutxline — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

getvfsaent — get vfstab_adjunct file entry 565, 567

getvfsafile — get vfstab_adjunct file entry 565, 567

group IDs, supplementary
initialize — initgroups 585

H

h_alloc — Allocate memory for a hexadecimal string 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

h_free — free hexadecimal string memory 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574

hextob — convert hexadecimal string to binary
 label 572, 577, 579, 581, 583
 htobcl — hexadecimal string to Binary CMW
 Label 572, 577, 579, 581, 583
 htobclear — hexadecimal string to Binary
 Clearance 572, 577, 579, 581,
 583
 htobil — hexadecimal string to Binary
 Information Label 572, 577,
 579, 581, 583
 htobsl — hexadecimal string to Binary
 Sensitivity Label 572, 577,
 579, 581, 583

I

initgroups — initialize the supplementary
 group access list 585

K

kernel virtual memory functions
 kstat_read — read or write kstat
 data 586–587
 kstat_write — read or write kstat
 data 586–587
 kstat_read — read or write kstat data 586–587
 kstat_write — read or write kstat data 586–587

L

bclearartos 158, 182, 207, 244, 262
 bcltos 158, 182, 207, 244, 262
 biltos 158, 182, 207, 244, 262
 bltocolr 240, 242
 bltocolr_r 240, 242
 bltype 248, 951
 bsltos 158, 182, 207, 244, 262
 sbclearartos 921, 924, 927, 930, 933
 sbcltos 921, 924, 927, 930, 933
 sbiltos 921, 924, 927, 930, 933
 sbsltos 921, 924, 927, 930, 933
 setbltype 248, 951
 stobcl 983, 988, 993, 998, 1003
 stobclear 983, 988, 993, 998, 1003
 stobil 983, 988, 993, 998, 1003
 stobsl 983, 988, 993, 998, 1003

tsol_lbuild_create() 588, 1194, 1200, 1206,
 1212
 tsol_lbuild_destroy() 588, 1194, 1200,
 1206, 1212
 tsol_lbuild_get() 588, 1194, 1200, 1206,
 1212
 tsol_lbuild_set() 588, 1194, 1200, 1206,
 1212
 Xbclearartos 594, 1233, 1235, 1237, 1239
 Xbcltos 594, 1233, 1235, 1237, 1239
 Xbiltos 594, 1233, 1235, 1237, 1239
 Xbsltos 594, 1233, 1235, 1237, 1239
 label library
 bclearhigh 148, 150, 162, 166, 168, 188,
 195, 197, 211, 231, 252, 254, 266
 bclearlow 148, 150, 162, 166, 168, 188, 195,
 197, 211, 231, 252, 254, 266
 bcleartoh 152, 155, 174, 177, 199, 202, 256,
 259, 270, 569, 574
 bcleartoh_r 152, 155, 174, 177, 199, 202,
 256, 259, 270, 569, 574
 bclearundef 148, 150, 162, 166, 168, 188,
 195, 197, 211, 231, 252, 254, 266
 bclearvalid 164, 213, 250, 268
 bclhigh 148, 150, 162, 166, 168, 188, 195,
 197, 211, 231, 252, 254, 266
 bcllow 148, 150, 162, 166, 168, 188, 195,
 197, 211, 231, 252, 254, 266
 bcltoh 152, 155, 174, 177, 199, 202, 256,
 259, 270, 569, 574
 bcltoh_r 152, 155, 174, 177, 199, 202, 256,
 259, 270, 569, 574
 bcltoil 180, 186, 205, 236, 500, 502, 953, 955
 bcltosl 180, 186, 205, 236, 500, 502, 953, 955
 bclundef 148, 150, 162, 166, 168, 188, 195,
 197, 211, 231, 252, 254, 266
 bildominates 191, 193, 215, 217, 221, 223,
 225, 227, 238
 bilequal 191, 193, 215, 217, 221, 223, 225,
 227, 238
 billow 148, 150, 162, 166, 168, 188, 195,
 197, 211, 231, 252, 254, 266
 biltoh 152, 155, 174, 177, 199, 202, 256, 259,
 270, 569, 574
 biltoh_r 152, 155, 174, 177, 199, 202, 256,
 259, 270, 569, 574

biltlev 180, 186, 205, 236, 500, 502, 953, 955
 bilundef 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266
 bimdominates 191, 193, 215, 217, 221, 223, 225, 227, 238
 bimequal 191, 193, 215, 217, 221, 223, 225, 227, 238
 bldominates 191, 193, 215, 217, 221, 223, 225, 227, 238
 blequal 191, 193, 215, 217, 221, 223, 225, 227, 238
 blinrange 191, 193, 215, 217, 221, 223, 225, 227, 238
 blmaximum 233–235
 blminimum 233–235
 blstrictdom 191, 193, 215, 217, 221, 223, 225, 227, 238
 bsllhigh 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266
 bslllow 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266
 bslltoh 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574
 bslltoh_r 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574
 bsllundef 148, 150, 162, 166, 168, 188, 195, 197, 211, 231, 252, 254, 266
 bsllvalid 164, 213, 250, 268
 getcil 180, 186, 205, 236, 500, 502, 953, 955
 getcsl 180, 186, 205, 236, 500, 502, 953, 955
 h_alloc 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574
 h_free 152, 155, 174, 177, 199, 202, 256, 259, 270, 569, 574
 htobcl 572, 577, 579, 581, 583
 htobclear 572, 577, 579, 581, 583
 htobil 572, 577, 579, 581, 583
 htobsl 572, 577, 579, 581, 583
 setcil 180, 186, 205, 236, 500, 502, 953, 955
 setcsl 180, 186, 205, 236, 500, 502, 953, 955
 bcltobanner
 bcltobanner 170
 bilconjoin 190
 blinset 229
 labelinfo 596
 labelvers 598

labelinfo — information about the label encodings 596
 labelvers — label_encodings file version 598
 library routines for client side calls
 — clnt_call 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — clnt_freeres 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — clnt_geterr 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — clnt_perrno 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — clnt_perror 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — clnt_sperrno 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — clnt_sperror 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — rpc_broadcast 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — rpc_broadcast_exp 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — rpc_call 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 — rpc_clnt_calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 library routines for dealing with creation and manipulation of CLIENT handles
 — clnt_control 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
 — clnt_create 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
 — clnt_create_timed 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
 — clnt_create_vers 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
 — clnt_create_vers_timed 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
 — clnt_destroy 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

- clnt_dg_create 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- clnt_pcreateerror 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- clnt_raw_create 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- clnt_spcreateerror 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- clnt_tli_create 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- clnt_tp_create 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- clnt_tp_create_timed 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- clnt_vc_create 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- rpc_clnt_create 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- rpc_createerr 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902

library routines for RPC servers

- rpc_svc_calls 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_dg_enablecache 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_done 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_exit 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_fdset 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_freeargs 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_getargs 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_getreq_common 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_getreq_poll 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_getreqset 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_getrpccaller 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_max_pollfd 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_pollfd 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_run 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- svc_sendreply 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119

libt6 — TSIX trusted IPC library 600

listen — listen for connections on a socket 605

lock address space

- mlockall 619, 623

lock memory pages

- mlock 617, 621

M

memory lock or unlock
 calling process — plock 772
memory management
 lock pages in memory — mlock 617, 619, 621, 623
 unlock pages in memory — munlock 617, 619, 621, 623
mldgetcwd — get pathname of current working directory 607
mldlstat — get status on symbolic link file in MLD 609, 615
mldrealpath — return absolute pathname 611, 613
mldrealpathl — return absolute pathname 611, 613
mldstat — get file status in MLD 609, 615

N

nftw — walk a file tree 445, 625
NIS+ table functions — nis_tables
 nis_first_entry 635, 655, 682, 705, 720, 740, 760
 nis_modify_entry 635, 655, 682, 705, 720, 740, 760
 nis_next_entry 635, 655, 682, 705, 720, 740, 760
 nis_remove_entry 635, 655, 682, 705, 720, 740, 760
NIS+ group manipulation functions
 — nis_addmember 644, 649, 652, 676, 679, 731, 749, 769
 — nis_creategroup 644, 649, 652, 676, 679, 731, 749, 769
 — nis_destroygroup 644, 649, 652, 676, 679, 731, 749, 769
 — nis_groups 644, 649, 652, 676, 679, 731, 749, 769
 — nis_ismember 644, 649, 652, 676, 679, 731, 749, 769
 — nis_print_group_entry 644, 649, 652, 676, 679, 731, 749, 769
 — nis_removemember 644, 649, 652, 676, 679, 731, 749, 769
 — nis_verifygroup 644, 649, 652, 676, 679, 731, 749, 769

NIS+ log administration functions

— nis_checkpoint 647, 729
— nis_ping 647, 729

NIS+ miscellaneous functions

— nis_freeresult 670, 672, 674, 697, 752, 754, 756, 758
— nis_frehtags 670, 672, 674, 697, 752, 754, 756, 758
— nis_getservlist 670, 672, 674, 697, 752, 754, 756, 758
— nis_mkdir 670, 672, 674, 697, 752, 754, 756, 758
— nis_rmdir 670, 672, 674, 697, 752, 754, 756, 758
— nis_server 670, 672, 674, 697, 752, 754, 756, 758
— nis_servstate 670, 672, 674, 697, 752, 754, 756, 758
— nis_stats 670, 672, 674, 697, 752, 754, 756, 758

NIS+ namespace functions

— nis_add 629, 664, 691, 699, 714, 734
— nis_freeresult 629, 664, 691, 699, 714, 734
— nis_lookup 629, 664, 691, 699, 714, 734
— nis_modify 629, 664, 691, 699, 714, 734
— nis_names 629, 664, 691, 699, 714, 734
— nis_remove 629, 664, 691, 699, 714, 734

NIS+ table functions

— nis_add_entry 635, 655, 682, 705, 720, 740, 760
— nis_first_entry 635, 655, 682, 705, 720, 740, 760
— nis_list 635, 655, 682, 705, 720, 740, 760
— nis_modify_entry 635, 655, 682, 705, 720, 740, 760
— nis_next_entry 635, 655, 682, 705, 720, 740, 760
— nis_remove_entry 635, 655, 682, 705, 720, 740, 760
— nis_tables 635, 655, 682, 705, 720, 740, 760

nis_tables — NIS+ table functions 635, 655, 682, 705, 720, 740, 760

nis_tables — NIS+ table functions 635, 655, 682, 705, 720, 740, 760

nis_tables — NIS+ table functions 635, 655,
682, 705, 720, 740, 760

P

plock — lock or unlock into memory process,
text, or data 772

processes

memory lock or unlock — plock 772

pututline — access utmp file entry 427, 536,
539, 542, 783, 976, 1226

pututxline — access utmpx file entry 430, 545,
549, 553, 557, 561, 786, 979,
1218, 1222, 1229

R

randomword

randomword 790

read or write kstat data

— kstat_read 586–587

— kstat_write 586–587

remote procedure calls, library routines for —
rpc 846

res_init — resolver routines 393, 399, 792, 798,
804, 810, 816, 822, 828, 834, 840

res_mkquery — resolver routines 393, 399,
792, 798, 804, 810, 816, 822,
828, 834, 840

res_mkupdate — resolver routines 393, 399,
792, 798, 804, 810, 816, 822,
828, 834, 840

res_mkupdrec — resolver routines 393, 399,
792, 798, 804, 810, 816, 822,
828, 834, 840

res_query — resolver routines 393, 399, 792,
798, 804, 810, 816, 822, 828,
834, 840

res_search — resolver routines 393, 399, 792,
798, 804, 810, 816, 822, 828,
834, 840

res_send — resolver routines 393, 399, 792,
798, 804, 810, 816, 822, 828,
834, 840

res_update — resolver routines 393, 399, 792,
798, 804, 810, 816, 822, 828,
834, 840

resolver — resolver routines 393, 399, 792, 798,
804, 810, 816, 822, 828, 834, 840

dn_comp 393, 399, 792, 798, 804, 810, 816,
822, 828, 834, 840

dn_expand 393, 399, 792, 798, 804, 810,
816, 822, 828, 834, 840

res_init 393, 399, 792, 798, 804, 810, 816,
822, 828, 834, 840

res_mkquery 393, 399, 792, 798, 804, 810,
816, 822, 828, 834, 840

res_mkupdate 393, 399, 792, 798, 804, 810,
816, 822, 828, 834, 840

res_mkupdrec 393, 399, 792, 798, 804, 810,
816, 822, 828, 834, 840

res_search 393, 399, 792, 798, 804, 810, 816,
822, 828, 834, 840

res_send 393, 399, 792, 798, 804, 810, 816,
822, 828, 834, 840

res_update 393, 399, 792, 798, 804, 810,
816, 822, 828, 834, 840

Return the extended credential information
associated with the client of
the current door invocation —
door_tcred 407

rpc — library routines for remote procedure
calls 846

RPC bind service library routines

— rpc_getmaps 856, 859, 862, 865, 868,
871, 882, 885

— rpcb_getaddr 856, 859, 862, 865, 868,
871, 882, 885

— rpcb_gettime 856, 859, 862, 865, 868,
871, 882, 885

— rpcb_rmtcall 856, 859, 862, 865, 868,
871, 882, 885

— rpcb_set 856, 859, 862, 865, 868, 871,
882, 885

— rpcb_unset 856, 859, 862, 865, 868, 871,
882, 885

— rpcbind 856, 859, 862, 865, 868, 871,
882, 885

RPC library routines for creation and
manipulation of server
handles

— rpc_svc_create 913, 1021, 1026, 1031,
1036, 1056, 1106, 1124, 1129,
1137

- `svc_create` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `svc_destroy` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `svc_dg_create` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `svc_fd_create` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `svc_raw_create` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `svc_tli_create` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `svc_tp_create` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `svc_vc_create` 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- RPC library routines for registering servers
 - `rpc_reg` 918, 1018, 1111, 1134, 1241, 1244
 - `rpc_svc_reg` 918, 1018, 1111, 1134, 1241, 1244
 - `svc_auth_reg` 918, 1018, 1111, 1134, 1241, 1244
 - `svc_reg` 918, 1018, 1111, 1134, 1241, 1244
 - `svc_unreg` 918, 1018, 1111, 1134, 1241, 1244
 - `xprt_register` 918, 1018, 1111, 1134, 1241, 1244
 - `xprt_unregister` 918, 1018, 1111, 1134, 1241, 1244
- `rpc_broadcast` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
- `rpc_broadcast_exp` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
- `rpc_call` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
- `rpc_clnt_calls` — library routines for client side calls 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
 - Routines 275, 321, 325, 335, 339, 355, 359, 874, 878, 888, 892
- `rpc_clnt_create` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
 - Routines 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- `rpc_createerr` — library routines for dealing with creation and manipulation of CLIENT handles 279, 285, 291, 297, 303, 309, 315, 329, 343, 349, 363, 369, 375, 381, 896, 902
- `rpc_svc_calls` — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
 - Routines 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
- `rpc_svc_create` — RPC library routines for creation and manipulation of server handles 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137
- `rpc_svc_reg` — RPC library routines for registering servers 918, 1018, 1111, 1134, 1241, 1244
- `rpcb_getaddr` — library routines for RPC bind service 856, 859, 862, 865, 868, 871, 882, 885
- `rpcb_getallmaps` — library routines for RPC bind service 856, 859, 862, 865, 868, 871, 882, 885
- `rpcb_getmaps` — library routines for RPC bind service 856, 859, 862, 865, 868, 871, 882, 885
- `rpcb_gettime` — library routines for RPC bind service 856, 859, 862, 865, 868, 871, 882, 885

rpcb_rmtcall — library routines for RPC bind
 service 856, 859, 862, 865,
 868, 871, 882, 885
 rpcb_set — library routines for RPC bind
 service 856, 859, 862, 865,
 868, 871, 882, 885
 rpcb_unset — library routines for RPC bind
 service 856, 859, 862, 865,
 868, 871, 882, 885
 rpcbind — library routines for RPC bind
 service 856, 859, 862, 865,
 868, 871, 882, 885

S

sbcleartos — Binary Clearance to canonical
 string 921, 924, 927, 930, 933
 sbcltos — Binary CMW Label to canonical
 string 921, 924, 927, 930, 933
 sbiltos — Binary Information Label to canonical
 string 921, 924, 927, 930, 933
 sbsltos — Binary Sensitivity Label to canonical
 string 921, 924, 927, 930, 933
 security policy 97
 send — send message from a socket 936, 938,
 940
 sendmsg — send message from a socket 936,
 938, 940
 sendto — send message from a socket 936,
 938, 940
 set_effective_priv — set the effective privileges
 for the current process. 957,
 959, 961
 set_inheritable_priv — set the inheritable
 privileges for the current
 process. 957, 959, 961
 set_permitted_priv — set the permitted
 privileges for the current
 process. 957, 959, 961
 setac — get audit control file information 409,
 449, 451, 453, 455, 457, 942
 setauclass — rewind audit_class database
 file 411, 459, 461, 463, 465,
 944
 setauuser — rewind audit_event database
 file 413, 473, 476, 479, 482,
 485, 488, 491, 946

setauuser — get audit_user database
 entry 416, 496, 498, 949
 setbltype — set type of Binary Label 248, 951
 setcil — replace Information Label 180, 186,
 205, 236, 500, 502, 953, 955
 setcsl — replace Sensitivity Label 180, 186,
 205, 236, 500, 502, 953, 955
 setprofent — sets the enumeration to the
 beginning of the set of Trusted
 Solaris profile entries 418,
 436, 511, 514, 963
 setsockopt — set options on sockets 523, 969
 setutent — access utmp file entry 427, 536,
 539, 542, 783, 976, 1226
 setutxent — access utmpx file entry 430, 545,
 549, 553, 557, 561, 786, 979,
 1218, 1222, 1229
 socket
 accept a connection — accept 120
 bind a name — bind 219
 get options — getsockopt 523, 969
 listen for connections — listen 605
 send message from — send, sendto,
 sendmsg 936, 938, 940
 set options — setsockopt 523, 969
 stobcl — string to Binary CMW Label 983, 988,
 993, 998, 1003
 stobclear — string to Binary Clearance 983,
 988, 993, 998, 1003
 stobil — string to Binary Information
 Label 983, 988, 993, 998, 1003
 stobsl — string to Binary Sensitivity Label 983,
 988, 993, 998, 1003
 str_to_auth — returns the numeric authorization
 ID 139, 141, 273, 434, 494,
 1008, 1010
 str_to_auth_set — breaks auth_names into
 tokens to be translated into an
 authorization list 139, 141,
 273, 434, 494, 1008, 1010

STREAMS

accept a connection on a socket —
 accept 120
 bind a name to a socket — bind 219
 get and set socket options — getsockopt,
 setsockopt 523, 969

listen for connections on a socket — listen 605	1086, 1091, 1096, 1101, 1114, 1119
send a message from a socket — send, sendto, sendmsg 936, 938, 940	
svc_auth_reg — RPC library routines for registering servers 918, 1018, 1111, 1134, 1241, 1244	svc_freeargs — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_control — RPC library routines for creation and manipulation of server handles 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137	svc_getargs — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_create — RPC library routines for creation and manipulation of server handles 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137	svc_getreq_common — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_destroy — RPC library routines for creation and manipulation of server handles 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137	svc_getreq_poll — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_dg_create — RPC library routines for creation and manipulation of server handles 913, 1021, 1026, 1031, 1036, 1056, 1106, 1124, 1129, 1137	svc_getreqset — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_dg_enablecache — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119	svc_gettrpcaller — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_done — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119	svc_max_pollfd — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_exit — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119	svc_pollfd — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081, 1086, 1091, 1096, 1101, 1114, 1119
svc_fdset — library routines for RPC servers 908, 1041, 1046, 1051, 1061, 1066, 1071, 1076, 1081,	svc_reg — RPC library routines for registering servers 918, 1018, 1111, 1134, 1241, 1244
	svc_run — library routines for RPC servers 908, 1041, 1046, 1051,

1061, 1066, 1071, 1076, 1081,
1086, 1091, 1096, 1101, 1114,
1119
 svc_sendreply — library routines for RPC
 servers 908, 1041, 1046, 1051,
 1061, 1066, 1071, 1076, 1081,
 1086, 1091, 1096, 1101, 1114,
 1119
 svc_tli_create — RPC library routines for
 creation and manipulation of
 server handles 913, 1021,
 1026, 1031, 1036, 1056, 1106,
 1124, 1129, 1137
 svc_tp_create — RPC library routines for
 creation and manipulation of
 server handles 913, 1021,
 1026, 1031, 1036, 1056, 1106,
 1124, 1129, 1137

T

t_accept — accept a connection request 1171
 t_bind — bind an address to a transport
 endpoint 1175
 t_optmgmt — manage options for a transport
 endpoint 1180
 t_snd — send data or expedited data over a
 connection 1187
 t_sndudata — send a data unit 1191
 t6alloc_blk — allocates security-attribute control
 structure and buffer 1142,
 1149
 t6attr_query — get mask indicating which
 attributes came from
 templates 1143
 t6clear_blk — clear security attributes 1144
 t6cmp_blk — compare security attributes 1145
 t6copy_blk — copy security attributes 1146
 t6dup_blk — duplicate security attributes 1147
 t6ext_attr — manipulate network-endpoint
 security options 1148, 1157
 t6free_blk — frees security-attribute control
 structure and buffer 1142,
 1149
 t6get_attr — get security attributes from the
 security-attribute buffer

handled by a control
 structure 1150, 1164
 t6get_endpt_default — get endpoint default
 attributes 1152, 1154, 1166,
 1168
 t6get_endpt_mask — get endpoint mask 1152,
 1154, 1166, 1168
 t6last_attr — examine the security attributes
 on the previous byte of
 data 1156, 1158
 t6new_attr — manipulate network-endpoint
 security options 1148, 1157
 t6peek_attr — examine the security attributes
 on the next byte of data 1156,
 1158
 t6recvfrom — read security attributes and data
 from a trusted endpoint 1159
 t6sendto — specify security attributes to
 send with data on a trusted
 endpoint 1161
 t6set_attr — set security attributes in the
 security-attribute buffer
 handled by a control
 structure 1150, 1164
 t6set_endpt_default — set endpoint default
 attributes 1152, 1154, 1166,
 1168
 t6set_endpt_mask — set endpoint mask 1152,
 1154, 1166, 1168
 t6size_attr — get the size of a particular
 attribute from the control
 structure 1170

Trusted Solaris profile entry

— endprofent 418, 436, 511, 514, 963
 — free_profent 418, 436, 511, 514, 963
 — getprofbyname 421, 439, 517, 520, 780,
 966
 — getprofent 418, 436, 511, 514, 963
 — getprofentbyname 418, 436, 511, 514,
 963
 — setprofent 418, 436, 511, 514, 963

Trusted Solaris profile strry

— endprofstr 421, 439, 517, 520, 780, 966
 — free_profstr 421, 439, 517, 520, 780, 966
 — getprofstr 421, 439, 517, 520, 780, 966
 — putprofstr 421, 439, 517, 520, 780, 966
 — setprofstr 421, 439, 517, 520, 780, 966

Trusted Solaris user entry
 — enduserent 424, 442, 527, 530, 533, 973
 — free_userent 424, 442, 527, 530, 533, 973
 — getuserbyname 424, 442, 527, 530, 533, 973
 — getuserbyuid 424, 442, 527, 530, 533, 973
 — getuserent 424, 442, 527, 530, 533, 973
 — setuserent 424, 442, 527, 530, 533, 973

U

unlock address space
 — munlockall 619, 623
 unlock memory pages
 — munlock 617, 621
 updwtmp — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229
 updwtmpx — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229
 user audit characteristics for socket or TLI peer
 get 506
 utmp file
 access entry — getutent 427, 536, 539, 542, 783, 976, 1226
 utmpname — access utmp file entry 427, 536, 539, 542, 783, 976, 1226
 utmpx file
 access entry — getutxent 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229
 utmpxname — access utmpx file entry 430, 545, 549, 553, 557, 561, 786, 979, 1218, 1222, 1229

V

vfstab_adjunct file
 — getvfsaent 565, 567

X

Xbclearos — Binary Clearance to string with
 font list 594, 1233, 1235, 1237, 1239
 Xbsltos — Binary CMW Label to string with
 font list 594, 1233, 1235, 1237, 1239
 Xbiltos — Binary Information Label to string
 with font list 594, 1233, 1235, 1237, 1239
 Xbsltos — Binary Sensitivity Label to string
 with font list 594, 1233, 1235, 1237, 1239
 XTSOLgetClientAttributes — get client
 attrs 1247
 XTSOLgetPropAttributes — get all prop
 attrs 1248
 XTSOLgetPropLabel — set resource label 1249
 XTSOLgetPropUID — get property uid 1250
 XTSOLgetResAttributes — get all attrs 1251
 XTSOLgetResLabel — get resource label 1252
 XTSOLgetResUID — set resource uid 1253
 XTSOLgetSSHheight — get screen stripe
 height 1254
 XTSOLgetWorkstationOwner — get ownership
 1255
 XTSOLIsWindowTrusted — test Trusted
 Window 1256
 XTSOLMakeTPWindow — Make Trusted path
 window 1257
 XTSOLsetPolyInstInfo — set poly instantiation
 info 1258
 XTSOLsetPropLabel — set resource label 1259
 XTSOLsetPropUID — set property uid 1261
 XTSOLsetResLabel — set resource label 1262
 XTSOLsetResUID — set resource uid 1263
 XTSOLsetSessionHI — set SL 1264
 XTSOLsetSessionLO — set SL 1265
 XTSOLsetSSHheight — set screen stripe
 height 1266
 XTSOLsetWorkstationOwner — set ownership
 1267
 XTSOLShutdown — shutdown system 1268