Copyright and publication information

This manual reflects version 4.2 of the OS-9 networking software.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Corporation.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.
# Contents

## Chapter 1: OS-9® Networking Structure

<table>
<thead>
<tr>
<th>Internet Data Module Structures</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostent</td>
<td>13</td>
</tr>
<tr>
<td>hostconfent</td>
<td>14</td>
</tr>
<tr>
<td>n_ifaliasreq</td>
<td>15</td>
</tr>
<tr>
<td>n_ifaliasreq6</td>
<td>16</td>
</tr>
<tr>
<td>n_ifnet</td>
<td>17</td>
</tr>
<tr>
<td>netent</td>
<td>19</td>
</tr>
<tr>
<td>protoent</td>
<td>20</td>
</tr>
<tr>
<td>resolvect</td>
<td>21</td>
</tr>
<tr>
<td>rtreq</td>
<td>22</td>
</tr>
<tr>
<td>rtreq6</td>
<td>23</td>
</tr>
<tr>
<td>servent</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point-to-Point Protocol Structures</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_handle</td>
<td>26</td>
</tr>
<tr>
<td>ppp_conninfo</td>
<td>27</td>
</tr>
<tr>
<td>ppp_error</td>
<td>28</td>
</tr>
<tr>
<td>ppp_hdlc_stats</td>
<td>29</td>
</tr>
<tr>
<td>ppp_modem_p</td>
<td>30</td>
</tr>
<tr>
<td>ppp_option_block</td>
<td>31</td>
</tr>
<tr>
<td>ppp_param_block</td>
<td>33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RPC/XDR Programming Structures</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth.h</td>
<td>35</td>
</tr>
<tr>
<td>clnt.h</td>
<td>36</td>
</tr>
<tr>
<td>svc.h</td>
<td>36</td>
</tr>
<tr>
<td>xdr.h</td>
<td>37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mbuf Data Structure</th>
<th>38</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbuf</td>
<td>38</td>
</tr>
</tbody>
</table>

## Chapter 2: Function Reference

<table>
<thead>
<tr>
<th>Internet Network Database Functions</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>netdb.l Library Functions</td>
<td>42</td>
</tr>
<tr>
<td>The Internet Library</td>
<td>42</td>
</tr>
<tr>
<td>delhostbyname()</td>
<td>45</td>
</tr>
<tr>
<td>delhostconfent()</td>
<td>46</td>
</tr>
<tr>
<td>delintbyname()</td>
<td>47</td>
</tr>
<tr>
<td>delresolvect()</td>
<td>48</td>
</tr>
<tr>
<td>delroutent()</td>
<td>49</td>
</tr>
<tr>
<td>delroutent6()</td>
<td>50</td>
</tr>
<tr>
<td>delservbyname()</td>
<td>51</td>
</tr>
<tr>
<td>endhostconfent()</td>
<td>52</td>
</tr>
<tr>
<td>endhostent()</td>
<td>53</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>pputoutent()</td>
<td>108</td>
</tr>
<tr>
<td>pputoutent6()</td>
<td>109</td>
</tr>
<tr>
<td>putservent()</td>
<td>110</td>
</tr>
<tr>
<td>res_cancel()</td>
<td>111</td>
</tr>
<tr>
<td>sethostent()</td>
<td>112</td>
</tr>
<tr>
<td>setnetent()</td>
<td>113</td>
</tr>
<tr>
<td>setprotoent()</td>
<td>114</td>
</tr>
<tr>
<td>setservent()</td>
<td>115</td>
</tr>
<tr>
<td>ndblib.l Library Functions</td>
<td>116</td>
</tr>
<tr>
<td>ndb_createNDbmod()</td>
<td>118</td>
</tr>
<tr>
<td>Berkeley Socket Functions</td>
<td>120</td>
</tr>
<tr>
<td>accept()</td>
<td>121</td>
</tr>
<tr>
<td>bind()</td>
<td>123</td>
</tr>
<tr>
<td>connect()</td>
<td>124</td>
</tr>
<tr>
<td>gethostname()</td>
<td>126</td>
</tr>
<tr>
<td>getpeermname()</td>
<td>127</td>
</tr>
<tr>
<td>getsockname()</td>
<td>128</td>
</tr>
<tr>
<td>getsockopt()</td>
<td>129</td>
</tr>
<tr>
<td>ip_start()</td>
<td>132</td>
</tr>
<tr>
<td>listen()</td>
<td>133</td>
</tr>
<tr>
<td>recv()</td>
<td>134</td>
</tr>
<tr>
<td>recvfrom()</td>
<td>136</td>
</tr>
<tr>
<td>recvmsg()</td>
<td>138</td>
</tr>
<tr>
<td>send()</td>
<td>139</td>
</tr>
<tr>
<td>sendto()</td>
<td>141</td>
</tr>
<tr>
<td>sendmsg()</td>
<td>143</td>
</tr>
<tr>
<td>sethostname()</td>
<td>144</td>
</tr>
<tr>
<td>setsockopt()</td>
<td>145</td>
</tr>
<tr>
<td>shutdown()</td>
<td>148</td>
</tr>
<tr>
<td>socket()</td>
<td>149</td>
</tr>
<tr>
<td>Point-to-Point Protocol Functions</td>
<td>152</td>
</tr>
<tr>
<td>ppp_auth_add_chap()</td>
<td>154</td>
</tr>
<tr>
<td>ppp_auth_add_pap()</td>
<td>156</td>
</tr>
<tr>
<td>ppp_auth_create_mod()</td>
<td>158</td>
</tr>
<tr>
<td>ppp_auth_del_chap()</td>
<td>159</td>
</tr>
<tr>
<td>ppp_auth_del_pap()</td>
<td>160</td>
</tr>
<tr>
<td>ppp_auth_get_cur_chap()</td>
<td>161</td>
</tr>
<tr>
<td>ppp_auth_get_cur_pap()</td>
<td>162</td>
</tr>
<tr>
<td>ppp_auth_get_peer_name()</td>
<td>163</td>
</tr>
<tr>
<td>ppp_auth_link_mod()</td>
<td>164</td>
</tr>
<tr>
<td>ppp_auth_set_peer_name()</td>
<td>165</td>
</tr>
<tr>
<td>ppp_auth_unlink_mod()</td>
<td>166</td>
</tr>
<tr>
<td>ppp_chat_close()</td>
<td>167</td>
</tr>
<tr>
<td>ppp_chat_open()</td>
<td>168</td>
</tr>
<tr>
<td>ppp_chat_read()</td>
<td>169</td>
</tr>
<tr>
<td>ppp_chat_script()</td>
<td>170</td>
</tr>
<tr>
<td>ppp_chat_write()</td>
<td>172</td>
</tr>
<tr>
<td>ppp_close()</td>
<td>173</td>
</tr>
<tr>
<td>ppp_connect()</td>
<td>174</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ITEM Library Functions</strong></td>
<td></td>
</tr>
<tr>
<td>ppp_disconnect()</td>
<td>176</td>
</tr>
<tr>
<td>ppp_get_asynch_params()</td>
<td>177</td>
</tr>
<tr>
<td>ppp_get_options()</td>
<td>178</td>
</tr>
<tr>
<td>ppp_get_params()</td>
<td>179</td>
</tr>
<tr>
<td>ppp_get_statistics()</td>
<td>180</td>
</tr>
<tr>
<td>ppp_init()</td>
<td>182</td>
</tr>
<tr>
<td>ppp_open()</td>
<td>183</td>
</tr>
<tr>
<td>ppp_reset_statistics()</td>
<td>184</td>
</tr>
<tr>
<td>ppp_set_asynch_params()</td>
<td>185</td>
</tr>
<tr>
<td>ppp_set_options()</td>
<td>187</td>
</tr>
<tr>
<td>ppp_start()</td>
<td>188</td>
</tr>
<tr>
<td>ppp_term()</td>
<td>189</td>
</tr>
<tr>
<td><strong>Per-Path Static Storage Functions</strong></td>
<td></td>
</tr>
<tr>
<td>pps_add_entry()</td>
<td>190</td>
</tr>
<tr>
<td>pps_chg_updrvr()</td>
<td>191</td>
</tr>
<tr>
<td>pps_del_entry()</td>
<td>192</td>
</tr>
<tr>
<td>pps_find_entry()</td>
<td>193</td>
</tr>
<tr>
<td>pps_find_entry_by_offset()</td>
<td>194</td>
</tr>
<tr>
<td><strong>ITEM Library Functions</strong></td>
<td></td>
</tr>
<tr>
<td>ite_ctl_addset()</td>
<td>196</td>
</tr>
<tr>
<td>ite_ctl_answer()</td>
<td>198</td>
</tr>
<tr>
<td>ite_ctl_connect()</td>
<td>200</td>
</tr>
<tr>
<td>ite_ctl_connstat()</td>
<td>202</td>
</tr>
<tr>
<td>ite_ctl_disconnect()</td>
<td>204</td>
</tr>
<tr>
<td>ite_ctl_rcvrasgn()</td>
<td>205</td>
</tr>
<tr>
<td>ite_ctl_rcvrrmv()</td>
<td>206</td>
</tr>
<tr>
<td>ite_data_avail_asgn()</td>
<td>208</td>
</tr>
<tr>
<td>ite_data_avail_rmv()</td>
<td>209</td>
</tr>
<tr>
<td>ite_data_read()</td>
<td>211</td>
</tr>
<tr>
<td>ite_data_readmbuf()</td>
<td>212</td>
</tr>
<tr>
<td>ite_data_readymbuf()</td>
<td>214</td>
</tr>
<tr>
<td>ite_data_read()</td>
<td>216</td>
</tr>
<tr>
<td>ite_data_recvfrom()</td>
<td>217</td>
</tr>
<tr>
<td>ite_data_sendro()</td>
<td>219</td>
</tr>
<tr>
<td>ite_data_write()</td>
<td>221</td>
</tr>
<tr>
<td>ite_data_writembuf()</td>
<td>223</td>
</tr>
<tr>
<td>ite_dev_attach()</td>
<td>224</td>
</tr>
<tr>
<td>ite_dev_detach()</td>
<td>225</td>
</tr>
<tr>
<td>ite_dev_getmode()</td>
<td>226</td>
</tr>
<tr>
<td>ite_dev_getname()</td>
<td>227</td>
</tr>
<tr>
<td>ite_dev_gettype()</td>
<td>228</td>
</tr>
<tr>
<td>ite_dev_setmode()</td>
<td>230</td>
</tr>
<tr>
<td>ite_fehangup_asgn()</td>
<td>231</td>
</tr>
<tr>
<td>ite_fehangup_rmv()</td>
<td>232</td>
</tr>
<tr>
<td>ite_linkdown_asgn()</td>
<td>233</td>
</tr>
<tr>
<td>ite_linkdown_rmv()</td>
<td>234</td>
</tr>
<tr>
<td>ite_linkup_asgn()</td>
<td>235</td>
</tr>
<tr>
<td>ite_linkup_rmv()</td>
<td>236</td>
</tr>
<tr>
<td>ite_path_clone()</td>
<td>237</td>
</tr>
<tr>
<td>ite_path_close()</td>
<td>239</td>
</tr>
</tbody>
</table>
Contents

RPC Functions .................................................................................................................. 280.
mbuf Functions................................................................................................................. 258.
Debugging Functions ........................................................................................................ 252.
Conv and OS Functions ................................................................................................. 249.
clnt_freeres() ................................................................................................................. 294.
clnt_destroy() .................................................................................................................. 293.
clnt_create().................................................................................................................. 292.
clnet_call() .................................................................................................................... 290.
clnet_broadcast() ........................................................................................................... 288.
callrpc() ........................................................................................................................ 286.
authnone_create().......................................................................................................... 283.
auth_destroy() ................................................................................................................ 282.
M_UNPAD(), M_UNPAD() .................................................................................................. 279.
m_pullup()..................................................................................................................... 277.
M_PREPEND() .................................................................................................................. 276.
m_pad(), M_PAD() .......................................................................................................... 274.
m_move() ....................................................................................................................... 272.
m_len_p() ....................................................................................................................... 271.
m_get() ........................................................................................................................... 269.
m_getn() ........................................................................................................................ 270.
m_free()......................................................................................................................... 268.
m_free_m() ..................................................................................................................... 266.
m_free_p()..................................................................................................................... 265.
m_free_q()..................................................................................................................... 264.
m_free()......................................................................................................................... 263.
m_free()......................................................................................................................... 262.
m_deq() ......................................................................................................................... 261.
m_cat()........................................................................................................................... 260.
m_copy() ........................................................................................................................ 260.
m_move()....................................................................................................................... 259.
m_adj()........................................................................................................................... 258.
RPC Functions .............................................................................................................. 280.
callrpc() ........................................................................................................................ 286.
clnet_broadcast()........................................................................................................... 288.
clnet_call() .................................................................................................................... 290.
clnet_control().............................................................................................................. 291.
clnet_create()................................................................................................................ 292.
clnet_destroy() ............................................................................................................. 293.
clnet_freeres()................................................................................................................ 294.

OS-9 Network Programming Reference 7
Contents

XDR Functions.................................................................................................................. 334
  xdr_accepted_reply()........................................................................................................ 337
  xdr_array().................................................................................................................... 338
  xdr_authunix_parms()................................................................................................. 339
  xdr_bool()..................................................................................................................... 340
  xdrbytes()..................................................................................................................... 341
  xdr_call HDR()............................................................................................................. 342
  xdr_callmsg()............................................................................................................. 343
  xdr_char()...................................................................................................................... 344
  xdr_destroy()............................................................................................................... 345
  xdr_double()................................................................................................................ 346
  xdr_free()..................................................................................................................... 346
  xdr_getmsg()................................................................................................................ 347
  xdr_getmsginfo()......................................................................................................... 348
  xdr_getmsglist()......................................................................................................... 349
  xdr_getmsglistinfo().................................................................................................... 349
  xdr_getmsglistmsg().................................................................................................... 350
  xdr_getmsglistmsginfo()............................................................................................. 351
  xdr_getmsglistmsglist().............................................................................................. 351
  xdr_getmsglistmsglistinfo()....................................................................................... 352
  xdr_getmsglistmsglistmsg()....................................................................................... 352
  xdr_getmsglistmsglistmsginfo()................................................................................ 353

svctcp_create()................................................................................................................ 332
svcraw_create()............................................................................................................... 331
svcerr_systemerr()........................................................................................................... 328
svcerr_noprog()................................................................................................................ 326
svcerr_noproc()................................................................................................................ 325
svcerr_decode()................................................................................................................ 324
svcerr_auth().................................................................................................................. 323
svc_unregister()........................................................................................................... 322
svc_sendreply()............................................................................................................. 321
svc_run()....................................................................................................................... 320
svc_register()................................................................................................................ 319
svc_getreqset()............................................................................................................. 318
svc_getcaller()............................................................................................................. 317
svc_getargs()................................................................................................................ 316
svc_freeargs()............................................................................................................... 315
registerrpc().................................................................................................................. 314
pmap_unset().................................................................................................................. 313
pmap_set()..................................................................................................................... 312
pmap_rmtcall()................................................................................................................ 311
get_myaddress()............................................................................................................ 310
svc_destroy()................................................................................................................ 309
svc_getargs()................................................................................................................ 308
svc_getcallers()............................................................................................................. 307
svc_getreqset()............................................................................................................. 306
svc_getreqsetmsg()....................................................................................................... 305
svc_getreqsetmsginfo()............................................................................................... 305
svc_getreqsetmsgmsg()............................................................................................... 304
svc_getreqsetmsgmsginfo()........................................................................................ 304
svc_getreqsetmsgmsgmsg()........................................................................................ 303
svc_getreqsetmsgmsgmsginfo()................................................................................... 303

clnt_geterr().................................................................................................................. 295
clnt_pcreateerror()....................................................................................................... 296
clnt_perorno()................................................................................................................. 297
clnt_perror().................................................................................................................. 298
clnt_spcreateerror()...................................................................................................... 299
clnt_sperrno()................................................................................................................ 300
clnt_sperror()............................................................................................................... 301
clnt_raw_create().......................................................................................................... 302
clnttcp_create()............................................................................................................ 303
clntudp_create()............................................................................................................ 304
get_myaddress()............................................................................................................ 305
pmap_getmaps()............................................................................................................ 306
pmap_getport()............................................................................................................. 307
pmap_rmtcall()............................................................................................................. 308
pmap_set()..................................................................................................................... 309
pmap_unset().................................................................................................................. 310
registerrpc()................................................................................................................ 311
svcc cancel().................................................................................................................. 312
svc_errno().................................................................................................................... 313
svc_errno().................................................................................................................... 314
svc_errno().................................................................................................................... 315
svc_errno().................................................................................................................... 316
svc_errno().................................................................................................................... 317
svc_errno().................................................................................................................... 318
svc_errno().................................................................................................................... 319
svc_errno().................................................................................................................... 320
svc_errno().................................................................................................................... 321
svc_errno().................................................................................................................... 322
svc_errno().................................................................................................................... 323
svc_errno().................................................................................................................... 324
svc_errno().................................................................................................................... 325
svc_errno().................................................................................................................... 326
svc_errno().................................................................................................................... 327
svc_errno().................................................................................................................... 328
svc_errno().................................................................................................................... 329
svc_errno().................................................................................................................... 330
svc_errno().................................................................................................................... 331
svc_errno().................................................................................................................... 332
svc_errno().................................................................................................................... 333

XDR Functions.................................................................................................................. 334
  xdr_accepted_reply()........................................................................................................ 337
  xdr_array().................................................................................................................... 338
  xdr_authunix_parms()................................................................................................. 339
  xdr_bool()..................................................................................................................... 340
  xdrbytes()..................................................................................................................... 341
  xdr_call HDR()............................................................................................................. 342
  xdr_callmsg()............................................................................................................. 343
  xdr_char()...................................................................................................................... 344
  xdr_destroy()............................................................................................................... 345
  xdr_double()................................................................................................................ 346
xdr_enum()..................................................................................................................... 347
xdr_float() ...................................................................................................................................... 348
xdr_free() ....................................................................................................................................... 349
xdr_getpos() ................................................................................................................................... 350
xdr_inline()................................................................................................................... .................. 351
xdr_int() ......................................................................................................................................... 352
xdr_long() ...................................................................................................................................... 353
xdr_opaque()................................................................................................................... ............... 354
xdr_opaque_auth() ......................................................................................................................... 355
xdr_pmap() .................................................................................................................................... 356
xdr_pmaplist()................................................................................................................. ............... 357
xdr_pointer() .................................................................................................................................. 358
xdr_reference() ............................................................................................................................... 359
xdr_rejected_reply()........................................................................................................... ............. 360
xdr_replmsg() ............................................................................................................................... 361
xdr_setpos()................................................................................................................... ................. 362
xdr_short() ..................................................................................................................................... 363
xdr_string() .................................................................................................................................... 364
xdr_u_char()................................................................................................................... ................ 365
xdr_u_int() ..................................................................................................................................... 366
xdr_u_long()................................................................................................................... ................ 367
xdr_u_short() ................................................................................................................................. 368
xdr_union() .................................................................................................................................... 369
xdr_vector()................................................................................................................... ................. 370
xdr_void() ...................................................................................................................................... 371
xdr_wrapstring() ............................................................................................................................ 372
xdrmem_create()............................................................................................................................. 373
xdrrec_create()................................................................................................................ ............... 374
xdrrec_endofrecord()........................................................................................................... ........... 375
xdrrec_eof()............................................................................................................................... 376
xdrrec_skiprecord() .................................................................................................................... 377
xprrt_register()................................................................................................................ ................. 378
xprrt_unregister()....................................................................................................................... 379
xrdtdio_create()................................................................................................................ ............... 380

Index
1

OS-9® Networking Structure

This chapter describes the OS-9® networking structures.

<table>
<thead>
<tr>
<th>For information about</th>
<th>Go to this page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Data Module Structures</td>
<td>12</td>
</tr>
<tr>
<td>Point-to-Point Protocol Structures</td>
<td>25</td>
</tr>
<tr>
<td>RPC/XDR Programming Structures</td>
<td>35</td>
</tr>
<tr>
<td>mbuf Data Structure</td>
<td>37</td>
</tr>
</tbody>
</table>
Internet Data Module Structures

`netdb.h` and `resolve.h` are header files that store information about Internet data module activities. The structures included in each of these header files are listed and described in the following table.

Table 1-1. `netdb.h` and `resolve.h` Structures and Descriptions

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hostent</code></td>
<td>Retrieve/Add/Remove Host Entry.</td>
</tr>
<tr>
<td><code>n_ifaliasreq</code></td>
<td>Add/Remove Interface Address Structure.</td>
</tr>
<tr>
<td><code>n_ifaliasreq6</code></td>
<td>Add/Remove IPV6 Interface Address Structure.</td>
</tr>
<tr>
<td><code>n_ifnet</code></td>
<td>Add Interface Structure.</td>
</tr>
<tr>
<td><code>netent</code></td>
<td>Retrieve/Add/Remove Network Entry.</td>
</tr>
<tr>
<td><code>protoent</code></td>
<td>Retrieve/Add/Remove Protocol Entry.</td>
</tr>
<tr>
<td><code>resolvent</code></td>
<td>Retrieve/Add/Remove DNS Client Entry.</td>
</tr>
<tr>
<td><code>rtreq</code></td>
<td>Add/Remove Route Structure.</td>
</tr>
<tr>
<td><code>rtreq6</code></td>
<td>Add/Remove Route Structure.</td>
</tr>
<tr>
<td><code>servent</code></td>
<td>Retrieve/Add/Remove Services Entry.</td>
</tr>
</tbody>
</table>
Declaration

The `hostent` structure is declared in the file `netdb.h` as follows:

```c
struct hostent {
    char *h_name;
    char **h_aliases;
    int h_addrtype;
    int h_length;
    char **h_addr_list;
#define h_addr h_addr_list[0]
};
```

Description

Returns and sets host entries in the Internet data module. This structure is used by the following functions:

- `gethostbyaddr()`
- `gethostbyname()`
- `puthostent()`
- `gethostent()`

Fields

- `h_name` is a pointer to the official name of the host.
- `h_aliases` points to a null-terminated array of pointers, which point to alternate names for the host.
- `h_addrtype` is the type of address (AF_INET only) returned.
- `h_length` is the address length in bytes.
- `h_addr` is a pointer to the network address for the host. Host addresses are returned in network-byte order.
- `h_addr_list` is a list of addresses from name server.
Declaration

The `hostconfent` structure is declared in the file `netdb.h` as follows:

```c
struct hostconfent {
    char *key;
    char *value;
};
```

Description

The `hostconfent` structure returns and sets host entries in the Internet data module. This structure is used by the following functions:

- `delhostconfent()`
- `endhostconfent()`
- `gethostconfent()`
- `puthostconfent()`

Fields

- `h_key` is a pointer to the key string.
- `h_value` is a pointer to value of the host.
n_ifaliasreq

Add/Remove Interface Address Structure

Declaration

The `n_ifaliasreq` is defined in `netdb.h` as follows:

```c
struct n_ifaliasreq {
    char ifra_name[16];
    struct sockaddr ifra_addr;
    struct sockaddr ifra_broadaddr;
    #define ifra_broadaddr ifra_dstaddr
    struct sockaddr ifra_mask;
} n_ifaliasreq;
```

Description

`n_ifaliasreq` is the interface structure for adding an interface. This structure is used in the function `putintent()`.

Fields

- `ifra_name` is the interface name.
- `ifra_addr` is the IP address.
- `ifra_broadaddr` is the broadcast address or destination address.
- `ifra_mask` is the subnet mask.
Declaration

The n_ifaliasreq6 is defined in netdb.h as follows:

```c
struct n_ifaliasreq6
{
    char    ifra_name[16];
    struct sockaddr_int6    ifra_addr;
    struct sockaddr_int6    ifra_dstaddr;
    struct sockaddr_int6    ifra_mask;
} n_ifaliasreq6;
```

Description

n_ifaliasreq6 is the interface structure for adding an Internet Protocol version 6 (IPv6) interface. This structure is used in the function putintent6().

Fields

- `ifra_name` is the IPv6 Interface name.
- `ifra_addr` is the IPv6 address.
- `ifra_broadaddr` is the IPv6 Broadcast address or destination address.
- `ifra_mask` is the IPv6 subnet mask.
Declaration

The `n_ifnet` structure is defined in `netdb.h` as follows:

```c
struct n_ifnet {
    char if_name[16];
    char if_stack_name[30];
    short if_flags;
    struct n_if_data {
        u_long ifi_type;
        u_long ifi_addrlen;
        u_long ifi_mtu;
        u_long ifi_metric;
    } if_data;
    u_long mw_flags;
} n_ifnet;
```

Description

The `n_ifnet` structure adds interface structures to the Internet data module.

Fields

- `if_name` is the interface name.
- `if_stack_name` is the binding device path.
- `if_flags` is the interface state:
  - Refer to `MWOS/SRC/DEFS/SPF/BSD/net/if.h` for possible values.
- `ifi_type` is the type of interface (not used, set to zero).
- `ifi_addrlen` is the hardware address length (not used; set to zero).
- `ifi_mtu` is the maximum transfer unit.
  - This is the largest piece of data handled by interface.
ifi_metric
    is the weighted routing metric.
    This is typically zero.

mw_flags
    is a Microware-defined interface flag.
    This is typically set to zero.

See Also
putintent()
Declaration

The netent structure is declared in the file netdb.h as follows:

```c
struct netent {
    char    *n_name;
    char    **n_aliases;
    int     n_addrtype;
    long    n_net;
};
```

Description

The `netent` structure obtains network entries in the Internet date module.

Fields

- **n_name**
  - is a pointer to the official name of the network.

- **n_aliases**
  - points to a null-terminated list of pointers, which points to alternate names for the network.

- **n_addrtype**
  - is the type of the network number (`AF_INET` only) returned.

- **n_net**
  - is the network number.

  Network numbers are returned in host-byte order.

See Also

- `getnetbyaddr()`
- `getnetbyname()`
- `getnetent()`
protoent
Retrieve/Add/Remove Protocol Entry

Declaration
The protoent structure is declared in the file netdb.h as follows:

```c
struct protoent {
    char *p_name;
    char **p_aliases;
    int p_proto;
};
```

Description
The protoent structure obtains protocol information in the Internet data module.

Fields
- **p_name**
  - is a pointer to the official name of the protocol.
- **p_aliases**
  - points to a null-terminated list of pointers, which points to alternate names for the protocol.
- **p_proto**
  - is the protocol number in host-byte order.

See Also
- getprotobyname()
- getprotobynumber()
- getprotoent()
Declaration

The `resolvent` structure is declared in the file `resolv.h` as follows:

```c
struct resolvent {
    char *domain;
    char *nameservers [MAXNS+1];
    char *search [MAXDNSRCH+1]
}
```

Description

The `resolvent` structure is used for updating and obtaining DNS client resolving information in the Internet data module.

Fields

domain

is a pointer to the local domain name.

nameservers

is an ordered list of `nameservers` (in dot notation).

search

is a search list for host-name lookup.

See Also

`getresolvent()`

`putresolvent()`
Declaration

The rtreq structure is declared in netdb.h as follows:

```c
struct rtreq {
    int req;
    int flags;
    struct sockaddr dst;
    struct sockaddr gateway;
    struct sockaddr netmask;
};
```

Description

The rtreq structure updates and obtains routing information in the Internet data module.

Fields

- **req**
  - is a request type (RTM_ADD).

- **flags**
  - is the type of route.
  - RTF_HOST for a host route.

- **dst**
  - is the destination address.

- **gateway**
  - is the gateway address.

- **netmask**
  - is the network mask address.

See Also

- putroutent()
- delroutent()

See the file MWOS/SRC/DEFS/SPF/BSD/net/route.h for additional req and flags settings.
Add/Remove Route Structure

Declaration

The `rtreq` structure is declared in `netdb.h` as follows:

```c
struct rtreq {
    int req;
    int flags;
    struct sockaddr_in6 dst;
    struct sockaddr_in6 gateway;
    struct sockaddr_in6 netmask;
};
```

Description

The `rtreq6` structure updates and obtains IPv6 routing information in the Internet data module.

Fields

- `req` is the request type (`RTM_ADD`).
- `flags` is the type of route.
  - `RTF_HOST` for a host route.
- `dst` is the IPv6 destination address.
- `gateway` is the IPv6 gateway address.
- `netmask` is the IPv6 network mask address.

See Also

- `putroutent6()`
- `delroutent()`

Refer to the file `MWOS/SRC/DEFS/SPF/BSD/net/route.h` for additional `req` and `flags` settings.
Declaration
The `servent` structure is declared in the file `netdb.h` as follows:

```c
struct servent {
    char    *s_name;
    char   **s_aliases;
    int    s_port;
    char    *s_proto;
};
```

Description
The `servent` structure obtains service information from the Internet data module.

Fields

- **s_name**
  is a pointer to the official name of the service.

- **s_aliases**
  points to a null-terminated list of pointers, which points to alternate names for the service.

- **s_port**
  is the port number at which the service resides.
  Port numbers are returned in network-byte order.

- **s_proto**
  is a pointer to the name of the protocol to use when contacting the service.

See Also

- `getservbyport()`
- `getservbyname()`
- `getservent()`
Point-to-Point Protocol Structures

There are five important structures that allow interaction between the calling application and the PPP stack. These structures are defined in the `ppplib.h` header file and are also listed below.

In all structures (except `auth_handle`), there are fields reserved for future use. It is important for the application to “zero” out all of these fields to ensure compatibility with future versions of drivers and this API. There are two methods to accomplish this: set the structure equal to zero or use the `memset()` function. The methods are described as follows:

```c
ppp_option_block     pppopts = {0};
```

or

```c
ppp_option_block     pppopts;
memset(&pppopts, 0, sizeof(pppopts));
```

Table 1-2 lists and describes the `ppplib.h` structures.

**Table 1-2. ppplib.h Structures and Descriptions**

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_handle</td>
<td>Allocated by Authentication Database Functions.</td>
</tr>
<tr>
<td>ppp_conninfo</td>
<td>Record PPP Connection Error Information</td>
</tr>
<tr>
<td>ppp_error</td>
<td>Store Error Information</td>
</tr>
<tr>
<td>ppp_hdlc_stats</td>
<td>Allow HDLC Driver to Return Current Receive and Transmit Statics.</td>
</tr>
<tr>
<td>ppp_modem_p</td>
<td>Get and Set Asynchronous Parameter or PPP Link</td>
</tr>
<tr>
<td>ppp_option_block</td>
<td>Obtain or Change Desired PPP Negotiation Options</td>
</tr>
<tr>
<td>ppp_param_block</td>
<td>Obtain Stack Parameters.</td>
</tr>
</tbody>
</table>
auth_handle

Allocated by Authentication Database Functions

Declaration

typedef struct _auth_handle
{
    mh_data       *mod_hdr;
    Auth_data     data;
}
auth_handle,*Auth_handle;

Description

The auth_handle structure is used by the authentication database functions. Applications do not have to interpret the internal portions of this structure (mod_hdr, data). However, they need to allocate auth_handle and pass a pointer to the authentication functions that fill out and manage the structure.
Declaration

```c
struct _ppp_conninfo{
    signal_code    sig_lcp_up;
    signal_code    sig_lcp_down;
    signal_code    sig_ipcp_up;
    signal_code    sig_ipcp_down;

    u_int32        flags;
    error_code     last_err;
    u_int32        max_errors;
    ppp_error      error_array;

    signal_code    sig_lcp_finish;
    signal_code    sig_ipcp_finish;
    u_int32        rsvd[4];
};

Ppp_conninfo, *Ppp_conninfo;
```

Description

The `ppp_conninfo` structure is allocated by the calling application and must be retained for the life of a PPP link. The `ppp_conninfo` area is where the PPP stack records error information regarding the PPP connection. Detailed error information is recorded in the `error_array`, which is a separately allocated array of `ppp_error` structures.

The following source code illustrates how to allocate a `ppp_conninfo` structure and details an error array that can hold `MAX_ERRORS` entries (error detection not shown):

```c
cpp_conninfo ci = {0};
u_int32 size;
size = sizeof(ppp_error) * MAX_ERRORS;
size = malloc(size);
memset(ci.error_array, 0, size);
```
Declaration

typedef struct _ppp_error {
    u_int32                layer;
    union {
        struct {
            error_code  err;
            u_int32     line;
            u_int32     abort_line;
        } chat;
        struct {
            u_int32     option;
            u_int32     my_request;
            u_int32     his_request;
        } lncp;
    } err_info;
} ppp_error, *Ppp_error;

Layer values may be PPP_LAYER_IPCP, PPP_LAYER_LCP, PPP_LAYER_HDLC, or PPP_LAYER_CHAT.

Description

When a significant error occurs within the PPP stack and an application has provided an error reporting array, information about the error will be stored in the first empty ppp_error slot. An empty slot is defined by the “layer” field having a value of 0.

The manner in which the information within the ppp_error structure is decoded depends on which layer recorded the error. For CHAT errors, the OS-9 error value is saved along with the offending CHAT line number. If an abort string has caused the CHAT script to terminate, the line number within the script, in which the abort string was defined, is also recorded. For LCP and IPCP layers, the desired option number, local request, and remote request are all recorded.
Allow HDLC Driver to Return Current Receive and Transmit Statics

Declaration

typedef struct _ppp_hdlc_stats{
  /*Receive Statistics */
  u_int32  rx_bytes;
  u_int32  rx_frames;
  u_int32  rx_frames_compressed;
  u_int32  rx_frames_dropped;
  u_int32  rx_frames_overrun;
  u_int32  rx_errors;
  u_int32  rx_fcs_errors;

  /*Transmit statistics */
  u_int32     tx_bytes;
  u_int32     tx_frames;
  u_int32     tx_frames_compressed;
  u_int32     tx_frames_dropped;
  u_int32     tx_frames_overrun;
  u_int32     tx_errors;
  
  u_int32     rsvd[3];
} ppp_hdlc_stats, *Ppp_hdlc_stats;

Description

The ppp_hdlc_stats structure is allocated by the calling application and referred to in the ppp_get_statistics() function. This structure allows the HDLC driver to return the current receive and transmit statics to the calling application.

For both the receive and transmit, the total number of bytes (bytes on the wire, including all HDLC framing bytes), errors (includes internal OS-9/SPF errors), and frames, including the Van-Jacobson compressed frames, dropped frames, and overrun frames (frames exceeding MTU/MRU) are part of this structure. For receive, there is also a field that tabulates the total number of frames having an invalid FCS (Frame Check Sequence); this is similar to a CRC check.
**Declaration**

```c
typedef struct ppp_modem_p{
    char      rx_dev_name[MAX_NAME_LEN];
    char      tx_dev_name[MAX_NAME_LEN];
    u_int8    baud_rate;
    u_int8    parity;
    u_int8    word_size;
    u_int8    stop_bits;
    u_int8    rts_enable;
    u_int32   rx_bufsize;
    u_int32   tx_bufsize;
} ppp_modem_p, *Ppp_modem_p;
```

**Description**

This structure is defined in ppp.h. It is used by the PPP function to get and set the asynchronous parameter or a PPP link, such as baud rate.
Declaration

typedef struct _ppp_option_block{
  /*Generic Stack Settings */
  u_int32     ppp_mode;

  /*IPCP-Specific Stack Options */
  u_int32     ipcp_timeout;
  u_int32     ipcp_max_configure;
  u_int32     ipcp_max.terminate;
  u_int32     ipcp_max_failure;
  struct      pppopt_ui32     ipcp_accept_local;
  struct      pppopt_ui32     ipcp_accept_remote;
  struct      pppopt_ui32     tx_ip_cproto;
  struct      pppopt_ui32     tx_ip_cproto;
  struct      pppopt_ui32     rx_ipcp_cslot;
  struct      pppopt_ui32     rx_ipcp_mslot;

  /*LCP-Specific Stack Options */
  u_int32     lcp_timeout;
  u_int32     lcp_max_configure;
  u_int32     lcp_max.terminate;
  u_int32     lcp_max_failure;
  struct      pppopt_ui32     rx_accm;
  struct      pppopt_ui32_8    tx_accm;
  struct      pppopt_ui32     rx_acfc;
  struct      pppopt_ui32     rx_pfc;
  struct      pppopt_ui32     rx_mru;
  struct      pppopt_ui32     tx_mru;
  struct      pppopt_ui32     auth_challenge;
  u_int32     ipcp_default_route;
  u_int32     rsvd[5];
} ppp_option_block, *Ppp_option_block;
Description

This structure is used to obtain or change the desired PPP negotiation options via the `ppp_get_options()` and `ppp_set_options()` functions, respectively. Many of these options are specified using a `pppopt_ui32` structure, which looks similar to the following code:

```c
struct pppopt_ui32
{
    u_int32   priority;/* DEFAULT, DESIRED, or REQUIRED */
    u_int32   value;
};
```

The `tx_acm` must have a size of 256-bits; therefore, it uses the following structure:

```c
struct pppopt_ui32_8
{
    u_int32   priority;
    u_int32   value[8];
};
```

These structures do not allow individual PPP options to be negotiated (DEFAULT), requested (DESIRED), or forced (REQUIRED), which causes a failure if the option cannot be negotiated. Priority levels are currently not supported and all stack options are treated as DESIRED. The “value” field depends on the option specified.

Most of these options are specified within the LCP and IPCP descriptors. The descriptor values will be used by default unless altered by a `ppp_set_options()` call.
typedef struct _ppp_param_block
{
    /*Generic stack parameters*/
    u_int32     ppp_mode;

    /*IPCP-specific stack parameters*/
    u_int32     rx_ip_cproto;
    u_int32     tx_ip_cproto;
    u_int32     rx_ipcp_cslot;
    u_int32     tx_ipcp_cslot;
    u_int32     rx_ipcp_mslot;
    u_int32     tx_ipcp_mslot;

    /*LCP-specific stack parameters*/
    u_int32     rx_accm;
    u_int32     tx_accm[8];
    u_int32     rx_acfc;
    u_int32     tx_acfc;
    u_int32     rx_pfc;
    u_int32     tx_pfc;
    u_int32     rx_mru;
    u_int32     tx_mru;
    u_int32     local_magic;
    u_int32     remote_magic;

    /*I/O enabled flags*/
    u_char     hdlc_io_enabled;
    u_char     lcp_io_enabled;
    u_char     ipcp_io_enabled;
    u_char     rsvd1;
    u_int32    rsvd2[3];
} ppp_param_block, *Ppp_param_block;
Description

The `ppp_param_block` structure is allocated by the calling application and used to obtain the stack parameters after LCP and IPCP negotiation has completed (both LCP and IPCP are I/O-enabled). Both transmit and receive parameters are returned at the same time.

Refer to the parameter descriptions for the `ppp_option_block` for more information on what each parameter represents.

The `local_magic` and `remote_magic` parameters are the negotiated "magic numbers" for the PPP link.

For more information on these parameters, refer to the `Request for Comment (RFC) 1661`. 
**RPC/XDR Programming Structures**

The main “include” file for all RPC files is `rpc.h`. It can be found in the following directory:

```
MWOS/SRC/DEFS/SPF/RPC
```

This header file includes the following RPC include files:

- RPC/xdr.h
- RPC/auth.h
- RPC/clnt.h
- RPC/svc.h
- RPC/pmap_clnt.h
- RPC/rpc_msg.h
- RPC/auth_unix.h
- RPC/svc_auth.h

### auth.h

The main structure in `RPC/auth.h` is shown below:

```c
/*
* Auth handle, interface to client side authenticators.
*/
typedef struct {
    struct opaque_auth  ah_cred;
    struct opaque_auth  ah_verf;
    struct auth_ops {
        void (*ah_nextverf)();
        int (*ah_marshall)();  /* nextverf & serialize */
        int (*ah_validate)();  /* validate varifier */
        int (*ah_refresh)();   /* refresh credentials */
        void (*ah_destroy)();  /* destroy this structure */
    } *ah_ops;
    caddr_t ah_private;
} AUTH;
```
clnt.h

The main structure in RPC/clnt.h is shown below:

/* Client rpc handle. Created by individual implementations, see e.g. rpc_udp.c. Client is responsible for initializing auth, see e.g. auth_none.c. */

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 */
    } *cl_ops;

    caddr_t cl_private; /* private stuff */
} CLIENT;

svc.h

The main structure RPC/svc.h is shown below:

* Server side transport handle */

typedef struct {
    int xp_sock;

    u_short xp_port; /* associated port number */

    struct xp_ops {
        bool_t (*xp_recv)(); /* receive incoming requests */
        enum xprt_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 address */

    struct sockaddr_in xp_raddr; /* remote address */

    struct opaque_auth xp_verf; /* raw response verifier */

    caddr_t xp_p1; /* private */

    caddr_t xp_p2; /* private */
} SVCXPRT;
xdr.h

The main structure in RPC/xdr.h is shown below:

/*
 * The XDR handle.
 * Contains operation which is being applied to the stream,
 * an operations vector for the particular implementation
 * (e.g. see xdr_mem.c),
 * and two private fields for the use of the particular implementation.
 */
typedef struct {
  enum xdr_op  x_op;  /* operation; fast additional param */
  struct xdr_ops {
    bool_t (*x_getlong)();  /* get a long from underlying stream */
    bool_t (*x_putlong)();  /* put a long to */
    bool_t (*x_getbytes)(); /* get some bytes from */
    bool_t (*x_putbytes)(); /* put some bytes to */
    u_int  (*x_getpostn)(); /* returns bytes off from beginning */
    bool_t (*x_setpostn)(); /* lets you reposition the stream */
    long * (*x_inline)();  /* buf quick ptr to buffered data */
    void  (*x_destroy)();  /* free privates of this xdr_stream */
  } *x_ops;
  caddr_t  x_public; /* users' data */
  caddr_t  x_private; /* pointer to private data */
  caddr_t  x_base; /* private used for position info */
  int      x_handy; /* extra private word */
} XDR;

mbuf Data Structure

The mbuf data structure is a common data structure used to store variable-length data blocks. The structure is defined below.
Declaration
The mbuf structure is declared in the mbuf.h header file as follows:

```c
typedef struct mbuf {
    struct mbuf *m_pnext;
    struct mbuf *m_qnext;
    u_int16 m_alloc;
    u_int16 m_size;
    u_int8  m_flags;
    u_int8  m_type;
    u_int16 m_offset;
} *mbuf;
```

Fields

- **m_pnext**: chains multiple mbufs into a packet chain called a message unit.

- **m_qnext**: chains multiple packet chains into a message queue.

- **m_alloc**: indicates the total size of the memory allocated for this mbuf. This is set by sysmbuf and must not be altered. If you do alter this field, system crashes result.

- **m_size**: indicates the number of valid data bytes in this mbuf.

- **m_flags**: contains bit flags which may be set to give special meaning to various mbufs. There are currently two uses for the m_flags field in an mbuf:

  1. Enable protocols to implement reliable data transfer through acknowledgment timeout and retransmission by using the SPF_NOFREE and SPF_DONE bits.

  2. Enable drivers to use protocols that contain receive packets with errors without tossing the entire packet.
m_type
indicates the type of mbuf.

The main use of the m_type field is to allow the network I/O system to
distinguish certain types of mbufs that may be on a data queue. The main
types used are MB_DATA and MB_ADDR. Others are defined but not widely used.

The MB_DATA type indicates the real payload that the end application wants to
read. The MB_ADDR indicates this mbuf is not the payload, but contains address
information the application may return when performing, for example, a
recvfrom() socket call.

Currently defined values are:
#define MB_NONE 0
#define MB_DATA 1
#define MB_ADDR 2
#define MB_HEADER 3

m_offset
indicates the offset in bytes, from the beginning of the mbuf to the first valid
bytes.

SPF_NOFREE/SPF_DONE

If you are writing a protocol driver that enables reliable data transfer, you typically
create a module array to store the pointers to data sent down for transmission, but
not yet acknowledged by the far end. However, when the driver is through
transmitting the packet, it typically performs an m_free_p() on the transmitted
mbuf. If this were allowed to happen, the mbuf in the unacknowledged array would
be lost.

The mbuf library implements the SPF_NOFREE to indicate that the mbuf must not be
returned to the free pool. When any m_free_x() call is done, the library checks for
the SPF_NOFREE bit. If it is set, the library does not return the mbuf to the free pool.
Instead, the library only sets the SPF_DONE bit to indicate that the packet has been
transmitted. With this approach, hardware drivers can still call the regular m_free()
functions and the library can take care of the SPF_NOFREE details.

There are special calls provided in misc.c that enable you to get and free what are
called “nofree” mbufs. These calls internally set and clear the SPF_NOFREE bit for
proper operation.
Each square in Figure 1-1 represents one mbuf structure.

The mbuf packet chain consists of the series of mbufs pointed to by the mbuf->m_pnext fields. In this figure, the mbuf packet chain for message #1 consists of three mbufs.

The mbuf queue consists of the series of mbuf packet chains pointed to by the mbuf->m_qnext fields. In this figure, the mbuf queue consists of messages one through four.
This chapter documents the OS-9 networking APIs.

<table>
<thead>
<tr>
<th>For information about...</th>
<th>Go to this page...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Network Database Functions</td>
<td>42</td>
</tr>
<tr>
<td>Berkeley Socket Functions</td>
<td>120</td>
</tr>
<tr>
<td>Point-to-Point Protocol Functions</td>
<td>152</td>
</tr>
<tr>
<td>Per-Path Static Storage Functions</td>
<td>190</td>
</tr>
<tr>
<td>ITEM Library Functions</td>
<td>196</td>
</tr>
<tr>
<td>Conv and OS Functions</td>
<td>249</td>
</tr>
<tr>
<td>Debugging Functions</td>
<td>252</td>
</tr>
<tr>
<td>mbuf Functions</td>
<td>258</td>
</tr>
<tr>
<td>RPC Functions</td>
<td>280</td>
</tr>
<tr>
<td>XDR Functions</td>
<td>334</td>
</tr>
</tbody>
</table>
Internet Network Database Functions

The following sections detail the network database functions.

netdb.l Library Functions

The Internet library (netdb.l) provides functions for retrieving information from the Internet data files or embedded in the inetdb data module and for Internet address manipulation. Each data access function links to inetdb and returns a structure pointing to the appropriate entry from the data files or modules.

There are two methods of linking to inetdb:

- Call sethostent(), setnetent(), setservent(), or setprotoent() explicitly links to inetdb.
- Call any of the Internet get functions implicitly links to inetdb.

To unlink a process from inetdb, use one of the end functions.

The Internet Library

LAN Communications provides two variations of the Internet library:

netdb.l This library provides bindings that call the shared netdb trap handler for OS-9 systems and subroutines for the OS-9 module.

netdb_dns.l This library contains functions for DNS client support and local hostname resolution. These functions do not call the netdb trap handler.

References to netdb.l also refer to netdb_dns.l in the function definitions.

netdb.l links to the netdb trap/subroutine module and executes the code in that module.

For application development, you can link with netdb_dns.l and create applications that do not call the netdb trap handler. The application size will be larger but the inlined code will execute faster. This may also remove the need for loading the netdb module.

To link with the netdb.l library, a netdb trap/subroutine module must be loaded prior to running the application.

netdb_dns The netdb_dns module provides Internet database functionality. To do this, it searches the local inetdb data module or uses the DNS client functionality to contact a name-server if a host name cannot be resolved or found locally within the inetdb data module.
The different combinations of library and trap/subroutine modules are identified in the following table.

### Table 2-1. Subroutine Modules

<table>
<thead>
<tr>
<th>Components</th>
<th>Shared Code via Trap Module</th>
<th>DNS Client Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>netdb.l + netdb_dns</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>netdb_dns.l</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The following table lists and describes the functions that compose the netdb.l libraries.

### Table 2-2. netdb.l Library Functions and Descriptions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delhostbyname()</td>
<td>Delete Host Entry by Name</td>
</tr>
<tr>
<td>delhostconfent()</td>
<td>Removes Host Entry by Key</td>
</tr>
<tr>
<td>delintbyname()</td>
<td>Delete Interface Entry by Name</td>
</tr>
<tr>
<td>delresolvent()</td>
<td>Delete DNS Resolver Entry</td>
</tr>
<tr>
<td>delroutent()</td>
<td>Delete Route Entry</td>
</tr>
<tr>
<td>delroutent6()</td>
<td>Delete IPV6 Route Entry</td>
</tr>
<tr>
<td>delservbyname()</td>
<td>Delete Service Entry by Name</td>
</tr>
<tr>
<td>endhostconfent()</td>
<td>Unlink from Structure</td>
</tr>
<tr>
<td>endhostent()</td>
<td>Unlink from Network Database</td>
</tr>
<tr>
<td>endintent()</td>
<td>Unlink from Network Database</td>
</tr>
<tr>
<td>endnetent()</td>
<td>Unlink from Network Database</td>
</tr>
<tr>
<td>endprotoent()</td>
<td>Unlink from Network Database</td>
</tr>
<tr>
<td>endresolvent()</td>
<td>Unlink from Network Database</td>
</tr>
<tr>
<td>endroutent()</td>
<td>Unlink from Network Database</td>
</tr>
<tr>
<td>endservent()</td>
<td>Unlink from Network Database</td>
</tr>
<tr>
<td>getaddrinfo()</td>
<td>Map Names to Address Information</td>
</tr>
<tr>
<td>gethostbyaddr()</td>
<td>Get Network Host Entry by Address</td>
</tr>
<tr>
<td>gethostbyname()</td>
<td>Get Network Host Entry by Name</td>
</tr>
<tr>
<td>gethostbyname2()</td>
<td>Get IPV4 or IPV6 Network Host Entry by Name</td>
</tr>
<tr>
<td>gethostconfent()</td>
<td>Get Network Entry by Name</td>
</tr>
<tr>
<td>gethostent()</td>
<td>Get Network Host Entry</td>
</tr>
<tr>
<td>getifaddrs()</td>
<td>Store List of Network Interfaces</td>
</tr>
<tr>
<td>getinetdent()</td>
<td>Get Inetd Entry</td>
</tr>
<tr>
<td>getintent()</td>
<td>Get Interface Entry</td>
</tr>
<tr>
<td>getnameinfo()</td>
<td>Look up IP Address and service name</td>
</tr>
<tr>
<td>getnetbyaddr()</td>
<td>Get Network Entry by Address</td>
</tr>
<tr>
<td>getnetbyname()</td>
<td>Get Network Entry by Name</td>
</tr>
<tr>
<td>getnetent()</td>
<td>Get Network Entry</td>
</tr>
<tr>
<td>getprotobynumber()</td>
<td>Get Protocol Entry by Number</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>getprotoent()</td>
<td>Get Protocol Entry</td>
</tr>
<tr>
<td>getresolvent()</td>
<td>Returns Pointer to DNS Structure</td>
</tr>
<tr>
<td>getservbyname()</td>
<td>Get Service Entry by Name</td>
</tr>
<tr>
<td>getservbyport()</td>
<td>Get Service Entry by Port</td>
</tr>
<tr>
<td>getservent()</td>
<td>Get Service Entry</td>
</tr>
<tr>
<td>htonl()</td>
<td>Convert 32-Bit Values from Host to Network Byte Order</td>
</tr>
<tr>
<td>htons()</td>
<td>Convert 16-Bit Values from Host to Network Byte Order</td>
</tr>
<tr>
<td>if_indextoname()</td>
<td>Map Interface Index to Interface Name</td>
</tr>
<tr>
<td>if_nameindex()</td>
<td>Return an Array of if_nameindex Structures</td>
</tr>
<tr>
<td>if_nametoindex()</td>
<td>Map Interface Name to Interface Index</td>
</tr>
<tr>
<td>inet_addr()</td>
<td>Convert Dot Notation into Network Address</td>
</tr>
<tr>
<td>inet_aton()</td>
<td>Convert Internet Address to Binary Address</td>
</tr>
<tr>
<td>inet_lnaof()</td>
<td>Get Local Address</td>
</tr>
<tr>
<td>inet_makeaddr()</td>
<td>Get Address from Network and Host Address</td>
</tr>
<tr>
<td>inet_netof()</td>
<td>Get Network Number</td>
</tr>
<tr>
<td>inet_network()</td>
<td>Interpret Network Number</td>
</tr>
<tr>
<td>inet_ntoa()</td>
<td>Return Address in Dot Notation</td>
</tr>
<tr>
<td>inet_ntop()</td>
<td>Convert network order to presentation order</td>
</tr>
<tr>
<td>inet_pton()</td>
<td>Convert presentation format address to network format</td>
</tr>
<tr>
<td>ntohl()</td>
<td>Convert 32-Bit Values from Network to Host Byte Order</td>
</tr>
<tr>
<td>ntohs()</td>
<td>Convert 32-Bit Values from Network to Host Byte Order</td>
</tr>
<tr>
<td>puthostent()</td>
<td>Add Host Entry</td>
</tr>
<tr>
<td>puthostconfent()</td>
<td>Add Network Host Entry</td>
</tr>
<tr>
<td>putintent()</td>
<td>Add Interface Entry</td>
</tr>
<tr>
<td>putintent6()</td>
<td>Add IPV6 Interface Entry</td>
</tr>
<tr>
<td>putnetent()</td>
<td>Add a network entry to inetdb</td>
</tr>
<tr>
<td>putprotoent()</td>
<td>Add protocol entry to inetdb</td>
</tr>
<tr>
<td>putresolvent()</td>
<td>Set the DNS Entry</td>
</tr>
<tr>
<td>putroutent()</td>
<td>Add Route Entry</td>
</tr>
<tr>
<td>putroutent6()</td>
<td>Add IPV6 Route Entry</td>
</tr>
<tr>
<td>putservent()</td>
<td>Add Interface entry to inetdb</td>
</tr>
<tr>
<td>res_cancel()</td>
<td>Cancel DNS Client Request</td>
</tr>
<tr>
<td>sethostent()</td>
<td>Set Host Entry</td>
</tr>
<tr>
<td>setnetent()</td>
<td>Set Network Entry</td>
</tr>
<tr>
<td>setprotoent()</td>
<td>Set Protocol Entry</td>
</tr>
<tr>
<td>setservent()</td>
<td>Set Services Entry</td>
</tr>
</tbody>
</table>
delhostbyname()
Delete Host Entry by Name

Syntax
#include <netdb.h>
error_code
delhostbyname(char *name)

Libraries
netdb.l

Description
delhostbyname() removes the host pointed to by name from the host section of the applicable inetdb data module. If successful, delhostbyname() returns 0; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.
EOS_PERMIT No permission.
EOS_PNNF Attempted to delete an entry that does not exist.
EOS_TRAP netdb module not found.
delhostconfent()
Removes Host Entry by Key

Syntax
#include <netdb.h>
error_code
delhostconfent (char *key)

Libraries
netdb.l

Description
delhostconfent() removes the host pointed to by key from the hostconfent structure. If successful, delhostconfent() returns 0; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_PARAM Bad parameter.
EOS_PNNF Attempted to delete an entry that does not exist.
EOS_PERMIT No permission.

See Also
hostconfent
endhostconfent()
gethostconfent()
puthostconfent()
delintbyname()
Delete Interface Entry by Name

Syntax
#include <netdb.h>
error_code
delintbyname(char *name)

Libraries
netdb.l

Description
delintbyname() removes the interface pointed to by name from the interface section of the inetdb data module. If successful, delintbyname() returns 0; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_PARAM Bad parameter.
EOS_PERMIT No permission.
EOS_PNNF Attempted to delete an entry that does not exist.
EOS_TRAP netdb module not found.
delresolvent()

Delete DNS Resolver Entry

Syntax
#include <netdb.h>
error_code
delresolvent(void)

Libraries
netdb.l

Description
delresolvent() removes the current DNS resolver entry from the appropriate
inetdb module. If successful, delresolvent() returns 0; otherwise it returns an
error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF netdb_dns module not found. This error is also returned
when the netdb_local module is loaded.
EOS_PARAM Bad parameter; not linked to an inetdb.
EOS_PNNF Attempted to delete entry that does not exist.
EOS_TRAP netdb module not found.

See Also
endresolvent() getresolvent() putresolvent()
Syntax
#include <netdb.h>
error_code
delroutent(const struct rtreq *route_ptr)

Libraries
netdb.l

Description
delroutent() removes the route entry matching route_ptr from the route section of the inetdb data module. Both the destination and gateway must match. If successful, delroutent() returns 0; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_PARAM Bad parameter.
EOS_PERMIT No write access.
EOS_PNNF Attempted to delete an entry that does not exist.
EOS_TRAP netdb module not found.

See Also
putroutent()
delroutent6()
Delete IPV6 Route Entry

Syntax
#include <netdb.h>
error_code
delroutent6(
    const struct rtreq6 *route_ptr)

Libraries
netdb.l

Description
delroutent6() removes the Internet Protocol version 6 (IPv6) route entry that matches route_ptr from the route section of the inetdb data module. Both the IPv6 destination and gateway must match. If successful, delroutent() returns 0; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_PARAM Bad parameter.
EOS_PERMIT No write access.
EOS_PNNF Attempted to delete an entry that does not exist.
EOS_TRAP netdb module not found.

See Also
putroutent6()
delservbyname()
Delete Service Entry by Name

Syntax

```c
error code delservbyname (
    char     *name)
```

Libraries

`netdb.l`

Description
delservbyname() removes the service entries pointed to by name from the `inetdb` data module. If successful, it returns 0; otherwise, it returns an error.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Errors

- EOS_PARAM Bad parameter.
- EOS_PNNF Attempted to delete an entry that does not exist.
- EOS_TRAP netdb module not found.
- EOS_PERMIT No write access.

See Also

`getservbyname()`
### Syntax

```c
#include <netdb.h>

error_code
endhostconfent (void)
```

### Libraries

`netdb.l`

### Description

`endhostconfent()` indicates the process is finished using the `hostconfent` structure. The link count of `inetdb` is decremented.

`endhostconfent()` returns 0 if successful; otherwise it returns an error.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Errors

- **ENOMEM** Insufficient RAM (POSIX).
- **EOS_PARAM** Bad parameter.

### See Also

- `hostconfent`
- `delhostconfent()`
- `gethostconfent()`
- `puthostconfent()`
Unlink from Network Database

Syntax
#include <netdb.h>
int endhostent(void)

Libraries
netdb.l

Description
endhostent() indicates the process is finished using the host section of the inetdb data module. The link count of inetdb is decremented.
endhostent() returns 0 if successful; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.

See Also
gethostbyaddr()
gethostbyname()
gethostent()
sethostname()
Syntax
#include <netdb.h>
error_code
endintent(void)

Libraries
netdb.l

Description
endintent() indicates the process is finished using the interface section of the inetdb data module. The link count of inetdb is decremented. If successful, endintent() returns 0; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.
endnetent()  
Unlink from Network Database

Syntax
#include <netdb.h>
int endnetent(void)

Libraries
netdb.l

Description
endnetent() indicates the process is finished using the network section of the
inetdb data module. The link count of inetdb is decremented. endnetent() returns
0 if successful; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.

See Also
getnetbyaddr()  
getnetbyname()  
getnetent()
endprotoent()

Unlink from Network Database

Syntax
#include <netdb.h>
int endprotoent(void)

Libraries
netdb.l

Description
endprotoent() indicates the process is finished using the protocol section of the inetdb data module. The link count of inetdb is decremented. endprotoent() returns 0 if successful; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.

See Also
getprotobyname()
getprotoent()
Unlink from Network Database

Syntax
#include <netdb.h>
endresolvent(void)

Libraries
netdb.l

Description
endresolvent() indicates the process is finished using the resolve section of the
inetdb data module. The link count of inetdb is decremented. endresolvent() returns 0 if successful; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.

See Also
getresolvent()
Syntax

#include <netdb.h>

error_code endroutent(void)

Libraries

netdb.l

Description

eindroutent() indicates the process is finished using the route section of the inetdb data module. The link count of inetdb is decremented. If successful, endroutent() returns 0.; otherwise it returns an error.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors

ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.
endservent()

Unlink from Network Database

Syntax
#include <netdb.h>
int endservent(void)

Libraries
netdb.l

Description
endservent() indicates the process is finished using the services section of the inetdb data module. The link count of inetdb is decremented. endservent() returns 0 if successful; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_PARAM Bad parameter.

See Also
getnetbyaddr()
getservbyname()
getservent()
setservent()
getaddrinfo()
Map Names to Address Information

Syntax

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(
    const char *nodename,
    const char *servname,
    const struct addrinfo *hints,
    struct addrinfo **res);
void freeaddrinfo(struct addrinfo *ai);
char *gai_strerror(int ecode);
```

Libraries

netdb.l

Description

getaddrinfo() provides a protocol independent way of mapping names to addresses information. Given a hostname and service name, the function searches from the beginning of the hosts and services entries of inetdb until a matching IP address and matching port number is found, or until EOF is encountered. If the entry is not in inetdb, it searches in DNS and returns either filled in addrinfo structure or error.

The addrinfo structure can be optionally passed as hints structure. The last four members must be set to zero or a NULL pointer.

All of the information returned by getaddrinfo() is dynamically allocated, freeaddrinfo() must be called to free the dynamically allocated memory.

Attributes

Operating System: OS-9
State: User
Threads: Not Safe
Parameters

nodename and servname
    pointers to null-terminated strings or NULL
    One or both of these must be a non-NULL pointer.

hints
    optional hints structure

res
    pointer to a linked list of one or more addrinfo structures

ai
    ai_addr member points to a filled-in socket address structure and ai_addrlen specifies its length.

Errors

The return value is 0 upon success or a nonzero error code. gai_strerror() can be used to print error messages based on the code returned by getaddrinfo() and defined in netdb.h.

See Also

getnameinfo()
gethostbyname()
gethostbyname2()
getservbyname()
gethostbyaddr()
Get Network Host Entry by Address

Syntax
#include <netdb.h>
struct hostent *gethostbyaddr(
    const char     *addr,
    int            len,
    int            type)

Libraries
netdb.l

Description
githubbyaddr() sequentially searches from the beginning of the hosts entries of
inetdb until a matching host address is found or until EOF is encountered. Host
addresses are supplied in network order. If an entry is not found in inetdb, the
search is done in DNS. The search order can be reversed by setting the
ENV_SERVORDER to “dns” or “bind”. (For example, search in DNS first; if nothing is
found, search in inetdb.)
githubbyaddr() returns a pointer to a hostent structure in the inetdb data
module. A null pointer (0) returns on EOF, or an error is returned and errno is set to
the error value.
githubbyaddr() implicitly links to inetdb if the calling process has not previously
linked to the data module.

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User
Threads:  Safe

Parameters
addr
    points to the Internet address of the host to get in network order.
len
    specifies the length of the address in bytes.
type
    specifies the AF_INET address type.

Errors
EOS_MNF  inetdb module could not be found.
See Also

gethostbyname()
gethostent()
sethostname()
gethostbyname()

Get Network Host Entry by Name

Syntax

```c
#include <netdb.h>
struct hostent *gethostbyname(const char *name)
```

Libraries

netdb.l

Description

gethostbyname() sequentially searches from the beginning of the hosts entries of inetdb until a matching host name or alias is found, or until EOF is encountered. If an entry is not found in inetdb, the search is done in DNS. The search order can be reversed by setting the ENV_SERVORDER to “dns” or “bind”. (For example, search in DNS first; if nothing is found, search in inetdb.)

gethostbyname() returns a pointer to a hostent structure in the inetdb data module. A null pointer (0) returns on EOF or error and errno is set to the error value.

gethostbyname() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

name

is a pointer to the name of the host.

Errors

EOS_MNF netdb module could not be found.

See Also

dendhostent()
gethostbyaddr()
gethostent()
gethostbyname2()
Get IPV4 or IPV6 Network Host Entry by Name

Syntax

```
#include <netdb.h>

struct hostent *gethostbyname2(const char *name, int af);
```

Libraries

```
netdb.l
```

Description

`gethostbyname2()` is an extension of `gethostbyname()` and it can also search for IPv6 network host entries. The second argument `af` must be set to `AF_INET` or `AF_INET6`. It sequentially searches from the beginning of the hosts entries of `inetdb` until a matching host name or alias is found, or until EOF is encountered. If an entry is not found in `inetdb`, the search is done in DNS. The search order can be reversed by setting the `ENV_SERVORDER` to “dns” or “bind”. (For example, serach in DNS first; if not found, search in inetdb.) `gethostbyname2()` returns a pointer to a hostent structure. A null pointer (0) returns on EOF or error and `errno` is set to the error value.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe

Parameters

- `name` points to the name of the host.
- `af` family type: `AF_INET` or `AF_INET6`

Errors

- `EOS_MNF` `netdb` module could not be found.

See Also

```
endhostent()
gethostbyname()
gethostbyaddr()
gethostent()
```
gethostconfent()
Reads Next Host Entry

Syntax
#include <netdb.h>
struct hostconfent *gethostconfent (void)

Libraries
netdb.l

Description
gethostconfent() reads the next host entry from inetdb. It returns a pointer to a hostent structure in the inetdb data module.
A null pointer returns an EOF or error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.
EOS_PARAM Bad parameter.

See Also
hostconfent
delhostconfent()
delhostconfent()
puthostconfent()
gethostent()

Get Network Host Entry

Syntax

#include <netdb.h>

struct hostent *gethostent(void)

Libraries

netdb.l

Description

gethostent() reads the next host entry from inetdb. It returns a pointer to a hostent structure in the inetdb data module. A null pointer returns on EOF or error. 
gethostent() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

See Also

donhostent()
gethostbyaddr()
gethostbyname()
sethostent()
getifaddrs()

Store List of Network Interfaces

Syntax
#include <sys/types.h>
#include <sys/socket.h>
#include <ifaddrs.h>

int getifaddrs(struct ifaddrs **ifap);
void freeifaddrs(struct ifaddrs *ifp);

Libraries
netdb.l

Description
The getifaddrs() function stores a reference to a linked list of the network interfaces on the local machine in the memory referenced by ifap. The list consists of ifaddr structures, as defined in the ifaddrs.h.

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

getifaddrs() allocates memory dynamically, and it should be freed using freeifaddrs().

Attributes
Operating System: OS-9
State: User
Threads: Safe

Parameters
ifap points to linked list of network interfaces.

See Also
if_indextoname()
if_nameindex()
if_nametoindex()
getinetdent()
Get Inetd Entry

Syntax
#include <netdb.h>
struct inetdent *getinetdent(void)

Libraries
netdb.l

Description
getinetdent() gets the next inetd entry. On success, getinetdent() returns a pointer to a inetdent structure in the inetdb module. A null pointer (0) returns on error and errno is set to the error value.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF netdb module could not be found.
EOS_PARAM Bad parameter.
getintent()  
Get Interface Entry

Syntax
#include <netdb.h>
char *getintent(void)

Libraries
netdb.l

Description
getintent() gets the next interface entry. On success, getintent() returns a pointer that is cast to a n_ifnet structure. Immediately following the n_ifnet structure is a u_int32, which indicates how many n_ifaliasreq structures follow it. Otherwise, it returns a null pointer (0) with the appropriate error code placed in the global variable errno.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF  netdb module could not be found.
EOS_PARAM  Bad parameter.
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(
    const struct sockaddr *sa,
    socklen_t salen,
    char *host,
    size_t hostlen,
    char *serv,
    size_t servlen,
    int flags);

Libraries
netdb.l

Description
getnameinfo() is a protocol-independent way of mapping address information to name. Given a IP address and port number, the function searches from the beginning of the hosts and services entries of inetdb until a matching host name or alias and matching port number (in network order) is found, or until EOF is encountered. If it does not find any entry in inetdb, it searches in DNS. Upon successful completion, it returns the hostname and service name in the buffer pointed to by the host and serv argument. The hostlen or servlen arguments must be set to zero value to specify not to return either string.

The flags can be set as shown below:

**NI_NOFQDN** By default, FQDN is searched and returned in DNS.
Setting this bit will return only the nodename portion of the FQDN of host.

**NI_NUMERICHOST** Return numeric form of host’s address instead of name.

**NI_NUMERICSERV** Return port number instead of port name.

**NI_DGRAM** This is a datagram service; getservport() is called with protocol set to udp instead of the default, tcp.

**NI_WITHSCOPEID** Return numeric IPv6 address notation with scope identifier.
getnameinfo()

Attributes
Operating System: OS-9
State: User
Threads: Not Safe

Parameters

sa
is a pointer to either a sockaddr_in (IPv4) or sockaddr_in6 (IPv6) structure that holds the IP address and port number.

salen
is the length of either sockaddr_in or sockaddr_in6 structure.

host
is a pointer to buffer for returned hostname.

hostlen
is the length of host buffer, defined as such in netdb.h:
#define NI_MAXHOST 1025

serv
is a pointer to buffer for returned service name.

servlen
is the length of serv buffer, defined as such in netdb.h:
#define NI_MAXSERV 32

Errors
The function returns zero upon successful completion, A non-zero value is returned upon failure.

See Also
getaddrinfo()
gethostbyaddr()
getservbyport()
getnetbyaddr()
Get Network Entry by Address

**Syntax**
```c
#include <netdb.h>
struct netent *getnetbyaddr(
    long     net,
    int      type)
```

**Libraries**
netdb.l

**Description**
getnetbyaddr() sequentially searches from the beginning of the networks entries of inetdb until a matching net address and type is found, or until EOF is encountered.
getnetbyaddr() returns a pointer to a netent structure in the inetdb data module. A null pointer (0) returns on EOF or error and errno is set to the error value.
getnetbyaddr() implicitly links to inetdb if the calling process has not previously linked to the data module.

**Attributes**
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

**Parameters**
net is the network number (network byte order).
type is the network number type (network byte order).

**Errors**
EOS_MNF netdb module could not be found.
EOS_PARAM Bad parameter.

**See Also**
endnetent()
getnetbyname()
getnetent()
setnetent()
getnetbyname()
Get Network Entry by Name

Syntax
#include <netdb.h>
struct netent *getnetbyname(const char *name)

Libraries
netdb.l

Description
getnetbyname() sequentially searches from the beginning of the networks entries of inetdb until a matching name or alias is found, or until EOF is encountered.
getnetbyname() returns a pointer to a netent structure in the inetdb data module. A null pointer (0) returns on EOF or error and errno is set to the error value.
getnetbyname() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
name is a pointer to the network name.

Errors
EOS_MNF  netdb module could not be found.
EOS_PARAM  Bad parameter.

See Also
ednetent()
getnetbyaddr()
getnetent()
setnetent()
getnetent()
Get Network Entry

Syntax

```c
#include <netdb.h>
struct netent *getnetent(void)
```

Libraries

netdb.l

Description

getnetent() reads the next inetdb network entry.
getnetent() returns a pointer to a netent structure in the inetdb data module. A null pointer (0) on EOF or error and errno is set to the error value.
getnetent() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors

EOS_MNF netdb module could not be found.
EOS_PARAM Bad parameter.

See Also

endnetent()
getnetbyaddr()
getnetbyname()
setnetent()
getprotobyname()
Get Protocol Entry

Syntax
#include <netdb.h>
struct protoent *getprotobyname(const char *name)

Libraries
netdb.l

Description
getprotobyname() sequentially searches from the beginning of the protocols entries of inetdb until it finds a matching protocol name or alias, or until it encounters EOF.

getprotobyname() returns a pointer to a protoent structure in the inetdb data module. getprotobyname() returns a null pointer (0) on EOF or error and places the error value in errno.

getprotobyname() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
name
is a pointer to the name of the protocol.

Errors
EOS_MNF netdb module could not be found.
EOS_PARAM Bad parameter.

See Also
endprotoent()
getprotobynumber()
getprotoent()
setprotoent()
getprotobynumber()
Get Protocol Entry by Number

Syntax
#include <netdb.h>
struct protoent *getprotobynumber(long proto)

Libraries
netdb.l

Description
getprotobynumber() sequentially searches from the beginning of the protocols entries of inetdb until it finds a matching protocol number, or until it encounters EOF.

getprotobynumber() returns a pointer to a protoent structure in the inetdb data module. getprotobynumber() returns a null pointer (0) on EOF or error and sets errno to the error value.

getprotobynumber() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
proto
specifies the protocol number in host byte order.

Errors
EOS_MNF inetdb module could not be found.
EOS_PARAM Bad parameter.

See Also
endprotoent()
getprotobyname()
getprotoent()
setprotoent()
getprotoent()
Get Protocol Entry

Syntax
#include <netdb.h>
struct protoent *getprotoent(void)

Libraries
netdb.l

Description
getprotoent() reads the next protocols entry of inetdb.
If successful, getprotoent() returns a pointer to a protoent structure in the inetdb data module. On EOF or error It returns a null pointer (0) and sets errno to the error value.
getprotoent() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.
EOS_PARAM Bad parameter.

See Also
endprotoent()
getprotobyname()
getprotobynumber()
setprotoent()
getresolvent()
Returns Pointer to DNS Structure

Syntax
#include <netdb.h>
struct resolvent *getresolvent(void)

Libraries
netdb.l

Description
getresolvent() returns a pointer to the resolver structure used by the DNS in the netdb data module. It returns the last one it finds. For example, the following modules are searched in this order (when x<4):

inetdbx
inetdbx-1
inetdb2
inetdb

The last valid pointer in the nameservers and search arrays is followed by a null pointer to indicate the end of valid data. On success, getresolvent() returns a pointer to a resolvent structure in the inetdb data module. On failure, it returns a null pointer and sets errno to the error value.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.
EOS_PARAM Bad parameter.

See Also
delresolvent()
endresolvent()
getservbyname()
Get Service Entry by Name

Syntax

#include <netdb.h>
struct servent *getservbyname(
    const char     *name,
    const char     *proto)

Libraries

netdb.l

Description

getservbyname() sequentially searches from the beginning of the services entries of inetdb until it finds a matching protocol name or alias, or until it encounters EOF. If a non-null protocol name is supplied, searches must also match the protocol.

getservbyname() returns a pointer to a servent structure in the inetdb data module. getservbyname() returns a null pointer (0) on EOF or error and sets errno to the error value.

getservbyname() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

name is a pointer to the name of the service.
proto is a pointer to the name of the protocol.

Errors

EOS_MNF inetdb module could not be found.
EOS_PARAM Bad parameter.

See Also

endservent()
getservbyport()
getservent()
setservent()
getservbyport()  
Get Service Entry by Port

Syntax

```c
#include <netdb.h>
struct servent *getservbyport(
    long           port,
    const char     *proto)
```

Libraries

netdb.l

Description

`getservbyport()` sequentially searches from the beginning of the services entries of `inetdb` until a matching protocol port number (in network order) is found, or until `EOF` is encountered. If a non-null protocol name is supplied, searches must also match the protocol.

`getservbyport()` returns a pointer to a `servent` structure in the `inetdb` data module. `getservbyport()` returns a null pointer (0) on `EOF` or error.

`getservbyport()` implicitly links to `inetdb` if the calling process has not previously linked to the data module.

Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

Parameters

- `port` specifies the service port number in network byte order.
- `proto` is a pointer to the protocol name.

Errors

- `EOS_MNF` `inetdb` module could not be found.
- `EOS_PARAM` Bad parameter.

See Also

- `endservent()`  
- `getservbyname()`  
- `getservent()`  
- `setservent()`
getservent()
Get Service Entry

Syntax
#include <netdb.h>
struct servent *getservent(void)

Libraries
netdb.l

Description
getservent() reads the next services entry of inetdb.
getservent() returns a pointer to a servent structure in the inetdb data module. It returns a null pointer (0) on EOF or error and sets errno to the error value.
getservent() implicitly links to inetdb if the calling process has not previously linked to the data module.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.
EOS_PARAM Bad parameter.

See Also
endservent()
getservbyname()
getservbyport()
setservent()
htonl()

Convert 32-Bit Values from Host to Network Byte Order

**Syntax**

```c
#include <sys/endian.h>
u_long htonl(u_long hostlong)
```

**Libraries**

netdb.l

**Description**

`htonl()` converts 32-bit quantities from host to network byte order and returns the long in a network byte order representation. `htonl()` is most often used with Internet addresses and ports as returned by `gethostent()` and `getservent()`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `hostlong` specifies the host byte order representation to convert.

**See Also**

- `htons()`
- `ntohl()`
- `ntohs()`
htons()  
Convert 16-Bit Values from Host to Network Byte Order

Syntax
#include <sys/endian.h>
int htons(u_short hostshort)

Libraries
netdb.l

Description
htons() converts 16-bit quantities from host to network byte order and returns the short in a network byte order representation. htons() is most often used with Internet addresses and ports as returned by gethostent() and getservent().

Attributes
Operating System:  OS-9 and OS-9 for 68K  
State:  User  
Threads:  Safe

Parameters
hostshort
    specifies the host byte order representation to convert.

See Also
htonl()  
nthol()  
nthohs()
**if_indextoname()**

Map Interface Index to Interface Name

**Syntax**

```c
#include <net/if.h>

char *if_indextoname(
    unsigned int ifindex,
    char *ifname);
```

**Libraries**

netdb.l

**Description**

`if_indextoname()` maps the interface index `ifindex` to its corresponding interface name, and returns it in buffer `ifname`, which must be of at least IFNAMSIZ bytes. If the specified interface does not exist, it returns 0.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `ifindex` is the interface index.
- `ifname` is a pointer to interface name, return value from function.

**Errors**

The function returns NULL pointer upon failure or if no interface is found.

**See Also**

- `getifaddrs()`
- `if_nameindex()`
- `if_nametoindex()`
if_nameindex()

Return an Array of if_nameindex Structures

Syntax
#include <net/if.h>

struct if_nameindex *if_nameindex(void);
void if_freenameindex(struct if_nameindex *ptr);

Libraries
netdb.l

Description
The if_nameindex() function returns an array of if_nameindex structures, one structure for each interface. The end of the array of structures is indicated by a structure with an if_index of 0 and an if_name of NULL. Upon successful completion, if_nametoindex() returns the index number of the interface.

The function allocates memory dynamically, freenameindex() must be called to free the memory allocated by if_nameindex(). freenameindex() takes argument returned by if_nameindex().

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Not Safe

Parameters
ptr

is a pointer to structure if_nameindex returned by if_nameindex.

Errors
The function returns NULL pointer upon failure or if no interface is found.

See Also
getifaddrs()
if_indextoname()
if_nametoindex()
if_nametoindex()  
Map Interface Name to Interface Index

Syntax
#include <net/if.h>
unsigned int if_nametoindex(const char *ifname);

Libraries
netdb.l

Description
The if_nametoindex() function maps the interface name specified in ifname to its corresponding index. If the specified interface does not exist, it returns 0.

Upon successful completion, if_nametoindex() returns the index number of the interface. A value of 0 is returned if the interface is not found or an error occurs while retrieving the list of interfaces via getifaddrs().

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Not Safe

Parameters
ifname is a pointer to interface name.

Errors
The function returns NULL pointer upon failure or if no interface is found.

See Also
getifaddrs()
if_indextoname()
if_nametoindex()
Syntax
#include <netdb.h>
unsigned long inet_addr(char *cp)

Libraries
netdb.l

Description
inet_addr() interprets character strings representing numbers expressed in the Internet standard “.” notation. It returns numbers suitable for use as Internet addresses, which are then returned in network order. On error, inet_addr() returns 0xFFFFFFFF (or -1 when cast to unsigned).

Refer to the Using LAN Communications manual for more information about Internet addresses.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
cp pointer to Internet address character string.

See Also
inet_aton()
inet_lnaof()
inet_makeaddr()
inet_netof()
inet_network()
inet_ntoa()
inet_aton()

Convert Internet Address to Binary Address

Syntax

```c
#include <netdb.h>
inet_aton(const char *cp, struct in_addr *addr)
```

Libraries

netdb.l

Description

inet_aton() determines whether the parameter `cp` is a valid ASCII representation of an Internet address and converts it to a binary address.

`inet_aton()` returns 1 if the address is valid and 0 if the address is not valid. This return value is more sophisticated than that of `inet_addr()`, which cannot distinguish whether the return is a failure or a broadcast address.

Refer to the Using LAN Communications manual for more information about Internet addresses.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `cp` is a pointer to the Internet address character string.
- `addr` is a pointer to the resulting Internet address.

See Also

- `inet_addr()`
- `inet_lnaof()`
- `inet_makeaddr()`
- `inet_netof()`
- `inet_network()`
- `inet_ntoa()`
inet_lnaof()

Get Local Address

Syntax
#include <netdb.h>
int inet_lnaof(struct in_addr in)

Libraries
netdb.l

Description
inet_lnaof() returns the host address portion of an Internet address (in host byte order). All host address parts are returned as integer values.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
in
    specifies the Internet address to break apart.

See Also
inet_addr()
inet_aton()
inet_makeaddr()
inet_netof()
inet_network()
inet_ntoa()
inet_makeaddr()

Get Address from Network and Host Address

Syntax

#include <sys/socket.h>
#include <netdb.h>
struct in_addr inet_makeaddr(
    int net,
    int lna)

Libraries

netdb.l

Description

inet_makeaddr() takes an Internet network number and a local network address and constructs an Internet address from it.

Refer to the Using LAN Communications manual for more information about Internet addresses.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

net
    specifies the network number in host byte order.

lna
    specifies the host number.

See Also

inet_addr()
inet_aton()
inet_lnaof()
inet_netof()
inet_network()
inet_ntoa()
inet_netof()

Get Network Number

Syntax
#include <netdb.h>
int inet_netof(struct in_addr in)

Libraries
netdb.l

Description
inet_netof() takes as input an Internet host address and returns the network number (in host byte order).

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
in
    specifies the Internet address.

See Also
inet_addr()
inet_aton()
inet_lnaof()
inet_makeaddr()
inet_network()
inet_ntoa()
Syntax
#include <netdb.h>
unsigned long inet_network(char *cp)

Libraries
netdb.l

Description
inet_network() interprets character strings representing numbers expressed in the Internet standard “.” notation. inet_network() returns numbers suitable for use as Internet network numbers. Network numbers are returned as unsigned long values (in host byte order).

On error, inet_network() returns 0xFFFFFFFF (or -1 when cast to unsigned).

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
cp
is a pointer to a character string.

See Also
inet_addr()
inet_aton()
inet_lnaof()
inet_makeaddr()
inet_netof()
inet_ntoa()
**inet_ntoa()**

Return Address in Dot Notation

**Syntax**

```c
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
char *inet_ntoa(struct in_addr in)
```

**Libraries**

`netdb.l`

**Description**

`inet_ntoa()` takes an Internet address and returns a pointer to a string in the Internet standard dot notation.

![Info Box]

Refer to the *Using LAN Communications* manual for more information about Internet addresses.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `in` specifies the Internet address to be converted to a string in network byte order.

**See Also**

- `inet_addr()`
- `inet_aton()`
- `inet_lnaof()`
- `inet_makeaddr()`
- `inet_netof()`
- `inet_network()`
inet_ntop()

Convert network order to presentation order

Syntax

```c
#include <sys/socket.h>
inet_ntop(int af,
         const void *src,
         char *dst,
         size_t size);
```

Libraries

netdb.l

Description

`inet_ntop()` converts a numeric address (network order) to presentation format. The returned IP address string is stored in buffer `dst`, which must be large enough to fit addresses. `size` is the size of the `dst` buffer. Following constants from `netinet/in.h` can be used to declare buffer of proper size for either IPV4 or IPV6 address.

```c
#define INET_ADDRSTRLEN 16
#define INET6_ADDRSTRLEN 46
```

The function returns pointer to IP address `dst` upon success or NULL on error.

Attributes

Operating System: OS-9
State: User
Threads: Safe

Parameters

- **af**
  - family type: `AF_INET` or `AF_INET6`
- **src** - a pointer to buffer holding IPv4 or IPv6 address in network order.
- **dst** - is a buffer for returned IP address.
- **size** - is the size of `dst`. 
Errors
NULL if error.

See Also
inet_aton()
inet_ntop()
inet_pton()
inet_addr()
inet_pton()
Convert presentation format address to network format

Syntax
#include <sys/socket.h>

int
inet_pton(
    int af,
    const char *src,
    void *dst);

Libraries
netdb.l

Description
inet_pton() converts an address from standard text presentation form to its
umeric binary form. The af argument specifies the family of the address. The
address is returned in network byte order. It returns 1 upon success, and 0 if input is
not a valid IPv4 dotted-decimal string or a valid IPv6 address string, and -1 if af
argument is unknown.

Attributes
Operating System: OS-9
State: User
Threads: Not Safe

Parameters
af
    family type: AF_INET or AF_INET6

src
    is a pointer to address string.

dst
    is a pointer to buffer for numeric IP address.
    The buffer must be large enough to hold numeric address: four bytes for
    AF_INET or 16 bytes for AF_INET6.

Errors
0 if input is not a valid IPv4 dotted-decimal string or a valid IPv6 address string,
and -1 if af argument is unknown.
See Also

inet_aton()
inet_ntop()
inet_addr()
ntohl()

Convert 32-Bit Values from Network to Host Byte Order

Syntax

```
#include <sys/endian.h>
u_long ntohl(u_long netlong)
```

Libraries

netdb.l

Description

ntohl() converts 32-bit quantities from network to host byte order and is used with Internet addresses and ports as returned by gethostent() and getservent().

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- netlong
  - specifies the 32-bit network byte order value to convert to host long.

See Also

- htonl()
- htons()
- ntohs()
ntohs()

Convert 32-Bit Values from Network to Host Byte Order

Syntax
#include <sys/endian.h>
u_short ntohs(u_short netshort)

Libraries
netdb.l

Description
ntohs() converts 16-bit quantities from network to host byte order. On machines such as the 68000, ntohs() is defined as a null macro in the include file in.h.

ntohs() is used with Internet addresses and ports as returned by gethostent() and getservent().

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
netshort
specifies the 16-bit network byte order value to convert to host short.

See Also
htonl()
htons()
ntohl()
puthostconfent()
Add Host Entry

Syntax
#include <netdb.h>
error_code
puthostconfent (char *key, char *value)

Libraries
netdb.l

Description
puthostconfent adds the host entry that is pointed to by the hostconfent structure to the inetdb data module (or an expansion inetdb data module). There must be space available in the data module to hold the host entry.

key and value are pointers to the hostconfent structure allocated and set by the user.
If successful, puthostconfent() returns 0; otherwise it returns an error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.
EOS_PARAM Bad parameter.
EOS_FULL No inetdb module found with enough space for the new entry.

See Also
hostconfent
delhostconfent ()
endhostconfent ()
gethostconfent ()
**puthostent()**

Add Network Host Entry

**Syntax**

```c
#include <netdb.h>
error_code puthostent(const struct hostent *hp)
```

**Libraries**

```text
netdb.l
```

**Description**

`puthostent()` adds the host entry pointed to by `hp` to the `inetdb` data module or an expansion `inetdb` data module. There must be space available in the data module to hold the host entry.

- `hp` is a pointer to a `hostent` structure allocated and set by the user.
- The `h_aliases` and `h_addr_list` elements are arrays of pointers to name and address information, and must be terminated by a null pointer.

If successful, `puthostent()` returns 0; otherwise it returns an error.

`puthostent()` supports a single IP address.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `hp` specifies the host entry to be added.

**Errors**

- **EOS_FULL** No `inetdb` module found with enough space for the new entry.
- **EOS_MNF** Not linked to a `netdb` module.
- **EOS_PARAM** Bad parameter.
- **EOS_TRAP** `netdb` module not found.

**See Also**

`gethostent()`
Syntax
#include <netdb.h>
error_code putintent(
    n_ifnet *ifp,
    n_ifaliasreq *ia,
    u_int32 ia_cnt)

Libraries
netdb.l

Description
putintent() adds an interface entry to inetdb data module or an expansion inetdb data module. There must be space available in the data module to hold the entry. If successful, putintent() returns 0.

The n_ifaliasreq and n_ifnet structures (defined in /netdb.h) are described in Chapter 1.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
ifp
    specifies a pointer to the user allocated and set interface structure.

ia
    specifies an array of null-terminated addresses to be associated with the interface.

ia_cnt
    specifies the number of addresses contained in the ia array, and may be 0.

Errors
EOS_MNF inetdb module not found.
EOS_FULL No inetdb module found with enough space for the new entry.

See Also
getnetent()
putintent6()
Add IPV6 Interface Entry

Syntax
#include <netdb.h>
error_code putintent6(
    n_ifnet *ifp,
    n_ifaliasreq6 *ia,
    u_int32 ia_cnt)

Libraries
netdb.l

Description
putintent6() adds an IPv6 interface entry to the inetdb data module or an expansion inetdb data module. There must be space available in the data module to hold the entry. If successful, putintent6 returns 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
ifp
    specifies a pointer to the user allocated and set interface structure.

ia
    specifies an array of null-terminated IPv6 addresses to be associated with the interface.

ia_cnt
    specifies the number of addresses contained in the ia array, and may be 0.

Errors
EOS_MNF
    inetdb module not found.

EOS_FULL
    No inetdb module found with enough space for the new entry.

See Also
getnetent()
putnetent()
Add a network entry to inetdb

Syntax
#include <netdb.h>
putnetent(const struct netent *np);

Libraries
netdb.l

Description
putnetent() adds a network entry to the inetdb data module or an expansion
inetdb data module. There must be space available in the data module to hold the
entry. If successful, putnetent returns 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
np specifies a pointer to the user allocated and initialized network entry
structure.

Errors
EOS_MNF inetdb module not found.
EOS_PARAM Bad parameter.
EOS_FULL No inetdb module found with enough space for the new
entry.
putprotoent()
Add protocol entry to inetdb

Syntax
#include <netdb.h>
putprotoent(const struct protoent *pep);

Libraries
netdb.l

Description
putprotoent() adds a protocol entry to the inetdb data module or an expansion inetdb data module. There must be space available in the data module to hold the entry. If successful, it returns 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
pep
specifies a pointer to the user allocated and initialized protocol entry structure.

Errors
EOS_MNF inetdb module not found.
EOS_PARAM Bad parameter.
EOS_FULL No inetdb module found with enough space for the new entry.
putresolvent()
Set the DNS Entry

Syntax
#include <resolv.h>
error_code putresolvent(resolvent *res)

Libraries
netdb.l

Description
putresolvent() adds the DNS resolver entry pointed to by res to the inetdb data module or an expansion inetdb data module. There must be space available in the data module to hold the resolver entry.

If successful, putresolvent() returns zero.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
res
is a pointer to a resolvent structure allocated and set by the user.

The name server and search pointer arrays must be terminated by a null pointer. There can be only one active resolver entry.

Only the first entry of the highest numbered inetdbx module is used. If you have enough space, you can add more than one, but it is ignored. To be safe, use delresolvent.

Errors
EOS_FULL  No inetdb module found with enough space for the new entry.
EOS_MNF  inetdb module not found.
EOS_PARAM  Bad parameter.
EOS_TRAP  netdb module not found.

See Also
delresolvent()
endresolvent()
getresolvent()
putroutent()
Add Route Entry

Syntax

```c
#include <netdb.h>
error_code *putroutent(struct rtreq *route_ptr)
```

Libraries

```
netdb.l
```

Description

`putroutent()` adds the route entry to the `inetdb` data module. If successful, `putroutent()` returns 0.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

Errors

- `EOS_MNF` - `inetdb` module not found.
- `EOS_PARAM` - Bad parameter.
- `EOS_FULL` - No `inetdb` module found with enough space for the new entry.
Syntax

```c
#include <netdb.h>
error_code *putroutent6(struct rtreq6 *route_ptr)
```

Libraries

`netdb.l`

Description

`putroutent6()` adds the IPv6 route entry to the `inetdb` data module. If successful, `putroutent()` returns 0.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_MNF</td>
<td><code>inetdb</code> module not found.</td>
</tr>
<tr>
<td>EOS_PARAM</td>
<td>Bad parameter.</td>
</tr>
<tr>
<td>EOS_FULL</td>
<td>No <code>inetdb</code> module found with enough space for the new entry.</td>
</tr>
</tbody>
</table>
putservent()
Add interface entry to inetdb

Syntax
#include <netdb.h>
putservent(const struct servent *sp)

Libraries
netdb.l

Description
putservent() adds an interface entry to the inetdb data module or an expansion
inetdb data module. There must be space available in the data module to hold the
entry. If successful, it returns 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
sp specifies a pointer to the user allocated and initialized interface entry
structure.

Errors
EOS_MNF inetdb module not found.
EOS_PARAM Bad parameter.
EOS_FULL No inetdb module found with enough space for the new entry.
res_cancel()

Cancel DNS Client Request

Syntax

```c
#include <netdb.h>
void res_cancel(void)
```

Libraries

netdb.l

Description

res_cancel() sets a flag to cancel a DNS client query (gethostbyname()). This stops an application from blocking on a gethostbyname() call. res_cancel() can only be used from the netdb_dns.l library since the trap library is not reentrant.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Syntax
#include <netdb.h>
int sethostent(int stayopen)

Libraries
netdb.l

Description
sethostent() links the calling process to inetdb, if necessary, and resets the pointer to the beginning of the hosts entries.
If successful, sethostent() returns a value of 0.
LAN Communications ignores the stayopen flag. It is included for compatibility only.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
ENOMEM Insufficient RAM (POSIX).
EOS_MNF inetdb module could not be found.

See Also
endhostent()
gethostbyaddr()
gethostbyname()
gethostent()
sethostname()
setnetent()
Set Network Entry

Syntax
#include <netdb.h>
int setnetent(int stayopen)

Libraries
netdb.l

Description
setnetent() links the calling process to inetdb, if necessary, and resets the pointer to the beginning of the network entries.

If successful, setnetent() returns a value of 0. Otherwise, it returns -1 and sets errno to the error value.

LAN Communications ignores the stayopen flag. It is included for compatibility only.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.

See Also
endnetent()
getnetent()
Syntax
#include <netdb.h>
int setprotoent(int stayopen)

Libraries
netdb.l

Description
setprotoent() links the calling process to inetdb, if necessary, and resets the pointer to the beginning of the protocol entries.
If successful, setprotoent() returns a value of 0. Otherwise, it returns -1 and sets errno to the error value.
LAN Communications ignores the stayopen flag. It is included for compatibility only.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.

See Also
getprotobynumber()
Syntax
#include <netdb.h>
int setservent(int stayopen)

Libraries
netdb.l

Description
setservent() links the calling process to inetdb, if necessary, and resets the pointer
to the beginning of the services entries.
If successful, setservent() returns a value of 0. Otherwise, it
returns -1 and sets errno to the error value.

LAN Communications ignores the stayopen flag. It is included for compatibility
only.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Errors
EOS_MNF inetdb module could not be found.

See Also
endservent()
getservbyname()
getservbyport()
getservent()
ndblib.l Library Functions

The ndblib.l library can be used to dynamically create an inetdb data module. This enables an application to configure a system before the IP stack is initialized.

The netdb.l library may call functions defined in ndblib.l. Therefore, when linking with netdb.l, ndblib.l must also be linked. The following table lists and describes the functions that compose the ndblib.l library.

Table 2-3. ndblib.l Functions and Descriptions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndb_create_ndbmod()</td>
<td>Create Network Database Module</td>
</tr>
</tbody>
</table>

Refer to the *Using LAN Communications* manual for more information about dynamic configuration.

Prototypes for these calls are declared in the following header files:

- MWOS/SRC/DEFS/SPF/BSD/sys/socket.h.
- MWOS/SRC/DEFS/SPF/BSD/netdb.h.
- MWOS/SRC/DEFS/SPF/BSD/netinet/in.h
- MWOS/SRC/DEFS/SPF/BSD/netinet6/in6.h

The sockaddr structure is defined in sys/socket.h as:

```c
struct sockaddr
{
    u_char sa_len;  /* total length */
    u_char sa_family;/* address family */
    char sa_data [14];/* up to 14 bytes of direct address */
};
```

This is a generic socket address meant to accompany various types of layer-three protocols. When using the IP protocol, only addresses belonging to the AF_INET or AF_INET6 protocol family are allowed.

The AF_INET type address is declared in the following file:

```c
struct sockaddr_in
{
    u_char sin_len;
    u_char sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char    sin_zero[8];
};
```
The in_addr structure is defined in the same file as shown below:

```c
struct in_addr
{
    u_long   s_addr;
};
```

The AF_INET6 type address is declared in the following header file:

```
MWOS/SRC/DEFS/SPF/bsd/netinet6/in6.h
```

It is defined as follows:

```c
struct sockaddr_in6 {
    u_int8_t   sin6_len;
    u_int8_t   sin6_family;
    u_int16_t   sin6_port;
    u_int32_t   sin6_flowinfo;
    struct in6_addr sin6_addr;
    u_int32_t   sin6_scope_id;
};
```

The in6_addr structure is defined in the same file as shown below:

```c
struct in6_addr{
    union {
        u_int8_t   __u6_addr8[16];
        u_int16_t   __u6_addr16[8];
        u_int32_t   __u6_addr32[4];
    } __u6_addr;
};
```
**ndb_create_ndbmod()**

Create Network Database Module

**Syntax**

```c
#include <netdblib.h>
error_code
ndb_create_ndbmod(
    char *modname,
    int num_files,
    int *file_sizes,
    u_int32 perm,
    u_int16 rev);
```

**Libraries**

ndblib.l

**Description**

`create_ndbmod` creates the Internet data module `modname` and reserves space as specified by the parameters passed.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **`modname`** is the name of the module to create.
  - While it can be any string, the `ndb_link_ndbmod` function (and hence the `netdb` calls) only check for `inetdb`, `inetdb2`, `inetdb3`, `inedb4`.

- **`num_files`** is the number of different files (record types) that this module stores.

File numbers 1-32 are assigned as shown in **Table 2-4**:

**Table 2-4. File Designations**

<table>
<thead>
<tr>
<th>File Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hosts (approx. 25 bytes per host)</td>
</tr>
<tr>
<td>2</td>
<td>hosts equiv (not used)</td>
</tr>
<tr>
<td>3</td>
<td>networks (approx. 40 bytes per network)</td>
</tr>
<tr>
<td>4</td>
<td>protocols (approx. 25 bytes per protocol)</td>
</tr>
<tr>
<td>5</td>
<td>services (approx. 25 bytes per service)</td>
</tr>
</tbody>
</table>
Table 2-4. File Designations (Continued)

| 6 | inetd entries (approx. 50 bytes per entry) |
| 7 | DNS client configuration (approx. 100 bytes) |
| 8 | local host configuration (not used) |
| 9 | host interfaces (approx 200 bytes per interface) |
| 10 | hostname (>= length of hostname + 1, recommended 65) |
| 11 | static routes (approx. 64 bytes per entry) |
| 12-32 | Reserved |

file_sizes

is an array of size num_files, where the value of element N indicates how many bytes to reserve for file N+1.

If file_sizes[6] = 400, then 400 bytes are reserved for storing resolver information.

perm

is the permission given to the newly created module.

If future updates are allowed, the user must have permission to link, read, and write to the module.

rev

The revision number given to the newly created module. An existing module may be overlaid if a higher revision number is specified.

Errors

EOS_PARAM

Bad parameter.
Berkeley Socket Functions

SPF includes an implementation of the Berkeley socket API in the socket.l library. The Internet socket library provides a BSD 4.4 socket API. The following table lists and describes the functions that compose the socket.l library:

Table 2-5. socket.l Functions and Descriptions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept()</td>
<td>Accept Connection on Socket</td>
</tr>
<tr>
<td>bind()</td>
<td>Binds Name to Socket</td>
</tr>
<tr>
<td>connect()</td>
<td>Initiates Connection on Socket</td>
</tr>
<tr>
<td>gethostname()</td>
<td>Gets Name of Current Host</td>
</tr>
<tr>
<td>getpeername()</td>
<td>Get Network Entry</td>
</tr>
<tr>
<td>getsockname()</td>
<td>Gets Socket Name</td>
</tr>
<tr>
<td>getsockopt()</td>
<td>Get Socket Options</td>
</tr>
<tr>
<td>ip_start()</td>
<td>Initialize IP stack</td>
</tr>
<tr>
<td>listen()</td>
<td>Listen for Connections on Socket</td>
</tr>
<tr>
<td>recv()</td>
<td>Receives Message From Connected Socket</td>
</tr>
<tr>
<td>recvfrom()</td>
<td>Receives Message from Socket</td>
</tr>
<tr>
<td>recvmsg()</td>
<td>Receives Message from Socket</td>
</tr>
<tr>
<td>send()</td>
<td>Sends Message to Connected Socket</td>
</tr>
<tr>
<td>sendto()</td>
<td>Sends Message to Socket</td>
</tr>
<tr>
<td>sendmsg()</td>
<td>Sends Message to Socket</td>
</tr>
<tr>
<td>sethostname()</td>
<td>Set Name of Current Host</td>
</tr>
<tr>
<td>setsockopt()</td>
<td>Set Options on Sockets</td>
</tr>
<tr>
<td>shutdown()</td>
<td>Shut Down Part of Full-Duplex Connection</td>
</tr>
<tr>
<td>socket()</td>
<td>Creates Endpoint for Communication</td>
</tr>
</tbody>
</table>

When creating socket applications, the ndblib.l and item.l libraries must also be linked.
Syntax
#include <sys/socket.h>
int accept(
    int s,
    struct sockaddr *addr,
    socklen_t *addrlen)

Libraries
socket.l
item.l (for 68K)

Description
accept() takes the first connection on the queue of pending connections and creates a new socket with the same properties as socket s. It allocates and returns a new socket descriptor. This new socket reads and writes data to and from the socket to which it is connected. It does not accept more connections. The original socket, s, remains open for accepting further connections.

If pending connections are nonexistent on the queue and the socket is configured for blocking, accept() blocks the caller until a connection is present.

If the socket is configured as non-blocking and pending connections are nonexistent on the queue, accept() returns EWOULDBLOCK.

accept() is used with connection-based socket types (SOCK_STREAM type only).
accept() returns -1 on error and errno is set to the error value. If successful, it returns a non-negative integer path number.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Parameters

\texttt{s}

is the original socket path.

It is created by \texttt{socket()}, bound to an address with \texttt{bind()}, and is listening for connections after \texttt{listen()}.

\texttt{addr}

is a pointer returning the address of the peer as known to the communications layer.

\texttt{addr}

is determined by the domain.

\texttt{addrlen}

is a pointer to a value-result parameter used to pass the amount of space pointed to by \texttt{addr}.

This parameter returns the actual length in bytes of the address returned.

Errors

\texttt{EINVAL}

The socket must be listening to call \texttt{accept()}.

\texttt{EOPNOTSUPP}

The referenced socket type or option is not supported.

\texttt{EWOULDBLOCK}

The socket is non-blocking and no connections are waiting to be accepted.

See Also

\texttt{bind()}
\texttt{connect()}
\texttt{listen()}
\texttt{socket()}

bind()

Binds Name to Socket

Syntax
#include <sys/socket.h>
int bind(
    int s,
    struct sockaddr *name,
    socklen_t namelen)

Libraries
socket.l

Description
bind() assigns a name to an unnamed socket. When socket() creates a socket, it exists in a name space (address family) but has no assigned name. bind() requests the name pointed to by the parameter name be assigned to the socket. Only names belonging to the AF_INET family are supported.

bind() returns 0 if successful. Otherwise, it returns -1 with the appropriate error code placed in the global variable errno.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
s
    specifies the path number of the socket.

name
    points to the socket address.

namelen
    specifies the length of the assigned name.

Errors
EADDRINUSE
    The specified address name is already in use.
EADDRNOTAVAIL
    The specified address name is not available on the local machine.
EINVAL
    The socket is already bound to an address name.
EOS_PERMIT
    The user is asking for a reserved port and the user is not super user.
connect()

Initiates Connection on Socket

Syntax

```c
#include <sys/socket.h>

int connect(
    int                     s,
    struct sockaddr        *name,
    socklen_t              namelen)
```

Libraries

socket.1

Description

If `s` is socket of type `SOCK_STREAM` (TCP), `connect()` attempts to connect to a listening socket. If the socket is a datagram socket such as `SOCK_DGRAM` (UDP) or `SOCK_RAW` (RAW), `connect()` stores the destination address locally. A successful connection returns 0. Otherwise, it returns -1 with the appropriate error code in `errno`.

If `s` is a non-blocking socket the initial `connect()` call returns `EINPROGRESS`. Subsequent calls return `EALREADY` until the connection is established, at which point an `EISCONN` error is returned.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `s` specifies the path number of the socket to connect. If `s` is of type `SOCK_STREAM`, `connect()` attempts to connect to another socket.
- `name` points to the other socket.
- `namelen` specifies the length of the assigned name.
Errors

EADDRINUSE  The address is already in use.
EADDRNOTAVAIL  The specified address is not available from this machine.
EAFNOSUPPORT  Address in the specified address family cannot be used with this socket.
EALREADY  The socket is non-blocking and a previous connection attempt has not been completed.
ECONNREFUSED  The attempt to connect was forcefully rejected.
EINPROGRESS  A non-blocking socket connection cannot be completed immediately.
EISCONN  The socket is already connected.
EHOSTONREACH  No route to the host exists
ETIMEDOUT  An attempt to establish a connection timed out without establishing the connection.

See Also

accept()
gethostname()

gets Name of Current Host

Syntax

```c
#include <sys/socket.h>
int gethostname(
    char *name,
    size_t namelen)
```

Libraries

socket.l

Description

gethostname() returns the host name for the current device. The returned name is
null-terminated string.

If successful, gethostname() returns a value of 0. Otherwise, it returns -1 and places
the appropriate error code in the global variable errno.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

name points to the standard host name.
namelen specifies the size of the name array.

Errors

EOS_MNF netdb module could not be found.
getpeername()
Get Network Entry

Syntax
#include <sys/socket.h>

int getpeername(
    int s,
    struct sockaddr *name,
    socklen_t *namelen
)

Libraries
socket.l

Description
getpeername() returns the name of the remote node (peer) connected to socket s. If successful, getpeername() returns 0. Otherwise, it returns -1 with errno set to the error value.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
s
    specifies the path number of the socket.

name
    is a pointer to the socket address.

namelen
    initializes namelen--before calling--to indicate the amount of space pointed to by name.

    On return, it contains the actual size of the name returned in bytes.

Errors
ENOTCONN       The socket is not connected.

See Also
bind()        getsockname()        socket()
getsockname()

Gets Socket Name

Syntax

```c
#include <sys/socket.h>

int getsockname(
    int s,
    struct sockaddr *name,
    socklen_t *namelen)
```

Libraries

socket.l

Description

getsockname() returns the current local node name for the specified socket. It returns 0 if the call succeeds. Otherwise, it returns -1 and sets errno to the error value.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `s` specifies the path number of the socket.
- `name` points to the socket address.
- `namelen` initializes namelen--before calling--to indicate the number of bytes pointed to by name. On return, it contains the actual size in bytes of the name returned.

Errors

- ENOBUFS Insufficient resources are available in the system to perform the operation.
- EBUFTOOSMALL Return buffer (name) is too small.

See Also

bind()
getpeercename()
socket()
getsockopt()

Get Socket Options

Syntax

```c
#include <sys/socket.h>

int getsockopt(
    int s,  
    int level,  
    int optname,  
    void* optval,  
    socklen_t* optlen)
```

Libraries

socket.l

Description

getsockopt() returns options associated with a socket. Options may exist at multiple protocol levels, but they are always present at the uppermost socket level. If successful, the call returns 0. Otherwise, it returns -1 and sets errno to the error value.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Options

socket.h contains definitions for the socket level options. Options at other protocol levels vary in format and name and are defined in the protocol’s header file.

Supported socket options are described in Table 2-6.

Table 2-6. Socket Level Options

<table>
<thead>
<tr>
<th>Level</th>
<th>optname</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPPROTO_IP</td>
<td>IP_MULTICAST_TTL</td>
<td>Get TTL for multicast packets</td>
<td>u_char</td>
</tr>
<tr>
<td>IPPROTO_IP</td>
<td>IP_MULTICAST_LOOP</td>
<td>Send multicast packets to the loopback interface</td>
<td>u_char</td>
</tr>
<tr>
<td>IPPROTO_IP</td>
<td>IP_MULTICAST_IF</td>
<td>Retrieve interface used for sending multicast packets</td>
<td>in_addr()</td>
</tr>
<tr>
<td>Level</td>
<td>optname</td>
<td>Description</td>
<td>Data Type</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>IPPROTO_TCP</td>
<td>TCP_MAXSEG</td>
<td>Get maximum segment size.</td>
<td>int</td>
</tr>
<tr>
<td>IPPROTO_TCP</td>
<td>TCP_NODELAY</td>
<td>Do not delay send to coalesce packets.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_KEEPALIVE</td>
<td>Keep connection alive by forcing peer to respond periodically.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_LINGER</td>
<td>Linger on close if data present.</td>
<td>linger()</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_REUSEADDR</td>
<td>Allow local address reuse.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_BROADCAST</td>
<td>Permit sending broadcast datagrams.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_OOBINLINE</td>
<td>Leave out-of-band data in normal input queue.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_SNDBUF</td>
<td>Get size of send buffer.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_RCVBUF</td>
<td>Get size of receive buffer.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_SNDLOWAT</td>
<td>Minimum space required in send buffer before to accept more data.</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_USELOOPBACK</td>
<td>Routing socket receives copy of any data sent (AF_ROUTE only).</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_TYPE</td>
<td>Get socket type (SOCK_STREAM, SOCK_DGRAM, SOCK_RAW).</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_ERROR</td>
<td>Retrieve pending socket error</td>
<td>int</td>
</tr>
</tbody>
</table>
Parameters

`s`

specifies the path number of the socket.

`level`

specifies the options level.

When getting socket options, you must specify the level at which the option resides and the option name.

To get options at the socket level, specify level as `SOL_SOCKET`.

To get options at any other level, supply the protocol number of the appropriate protocol controlling the option. For example, `IPPROTO_TCP` specifies TCP options.

`optname`

specifies the name of the option. optname and any specified options are passed uninterpreted to the appropriate protocol module.

`optval`

is a pointer to the buffer for the requested option.

`optval` and `optlen` together identify the buffer in which to return the value for the requested option(s). If an option value is not to be supplied or returned, set `optval` to 0.

`optlen`

is a pointer to a value-result parameter.

`optlen` initially contains the size of the buffer pointed to by `optval`. `optlen` is modified on return to indicate the actual size of the returned value.

Errors

`ENOPROTOOPT` The option is unknown.

See Also

`setsockopt()`

`socket()`
ip_start()

Initialize IP stack

Syntax

```c
#include <sys/socket.h>
error_code ip_start(void)
```

Libraries

socket.l

Description

`ip_start()` initializes the IP stack and all the configured drivers. If successful, `ip_start()` returns 0. Otherwise, it returns the appropriate error code.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Errors

- EOS_MNF Module not found.
listen()

Listen for Connections on Socket

Syntax
#include <sys/socket.h>

int listen(
    int s,
    int backlog)

Libraries
socket.l

Description

listen() marks an existing socket as willing to accept incoming connections. The incoming connections are queued until accept() is called to retrieve them. The listen call applies only to sockets of type SOCK_STREAM.

listen() returns 0 if successful. Otherwise, it returns a -1 with the appropriate code in errno.

Attributes

Operating System:  OS-9 and OS-9 for 68K
State:  User
Threads:  Safe

Parameters

s
specifies the path number of the socket.

backlog
define maximum length to which the queue of pending connections may grow.

The backlog is limited from 0 to the maximum queue of 128. If the backlog is greater than 128, it defaults to 128. If a connection request arrives with the queue full, the client receives an ECONNREFUSED error.

Errors

EINVAL  The socket must be bound in order to listen.
EOPNOTSUPP  The socket is not of a type supporting the operation listen().

See Also

accept()
connect()
socket()
recv()

Receives Message From Connected Socket

Syntax

```c
#include <sys/socket.h>
size_t recv(
    int     s,
    void    *buf,
    size_t  len,
    int     flags);
```

Libraries

socket.l

Description

recv() receives messages from a socket. Use recv() only on connected sockets.
If no data are available at the socket, the call waits for data to arrive, unless the
socket is non-blocking. In this case, -1 is returned with the external variable errno
set to EWOULDBLOCK.

On success, the number of bytes read is returned. This value may be less than the
number of bytes requested in len. If the socket is of type SOCK_STREAM, a return
value of 0 indicates the peer has closed its half of the connection and no more data
is available to read. On error, -1 is returned and errno is set to the error value.

Attributes

Operating System:     OS-9 and OS-9 for 68K
State:                User
Threads:              Safe

Parameters

s
    specifies the path number of the socket.

buf
    points to the buffer into which the message is received.

len
    specifies the length of the buffer.

flags
    MSG_PEEK; MSG_WAITALL
**Errors**

- **ENOTCONN**: The socket is not connected.
- **EWOULDBLOCK**: The socket is marked non-blocking and the receive operation would block.
**Syntax**

```c
#include <sys/socket.h>
ssize_t recvfrom(
    int               s,
    void              *buf,
    size_t            len,
    int               flags,
    struct sockaddr   *from,
    socklen_t         *fromlen);
```

**Libraries**

socket.l

**Description**

`recvfrom()` receives messages from a socket. You may use `recvfrom()` to receive data on a socket in an unconnected state.

`recvfrom()` returns the length of the incoming message. 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.

If no data are available at the socket, the call waits for data to arrive, unless the socket is non-blocking. In this case, -1 is returned with the external variable `errno` set to `EWOULDBLOCK`.

On success the number of bytes read is returned. This value may be less than the number of bytes requested in `len`. If the socket is of type `SOCK_STREAM`, a return value of 0 indicates the peer has closed its half of the connection and no more data is available to read. On error, -1 is returned and `errno` is set to the error value.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `s` specifies the path number of the socket.
- `buf` points to the buffer into which the message is received.
recvfrom()

len
specifies the length of the buffer.

flags
MSG_PEEK; MSG_WAITALL.

from
points to a buffer specifying the sender of the message. If from is non-zero, the
source address of the message is filled in.

fromlen
is initialized to the size of the buffer associated with from. fromlen is modified
on return to indicate the actual size of the address stored.

Errors

EWOULDBLOCK The socket is marked non-blocking and the receive operation
is blocked.
recvmsg()

Receive Messages from Socket

Syntax
#include <sys/socket.h>
ssize_t recvmsg (ints,
   struct msghdr *msg,
   int flags);

Libraries
socket.l

Description
recvmsg() uses an msghdr structure to minimize the number of directly supplied parameters.

Attributes
Operating System: OS-9
State: User
Threads: Safe

Parameters

s
   specifies the path number of the socket.

msg
   points to a message header structure defined in <sys/socket.h>

flags
   MSG_PEEK; MSG_WAITALL

Errors
EWOULDBLOCK  The socket is marked non-blocking and the receive operation is blocked.
send()

Sends Message to Connected Socket

Syntax

```c
#include <sys/socket.h>
ssize_t send(
    int s,
    const void* msg,
    size_t len,
    int flags);
```

Libraries

socket.l

Description

`send()` transmits a message to another socket. Use `send()` only when the socket is in a connected state.

If the socket is a datagram socket and the message is too long to pass atomically through the underlying protocol, an error is returned and the message is not transmitted.

For reliable protocols, such as TCP, if no message space is available at the socket to hold the transmitted message, `send()` normally blocks, unless the socket has been placed in non-blocking I/O mode.

No indication of failure to deliver is implicit in a send. Return values of `-1` indicate some locally detected errors. On success, `send()` returns the number of characters sent. On error, `-1` is returned and `errno` is set to the error value.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `s` specifies the path number of a socket created with `socket()`.
- `msg` points to the message to send.
- `len` specifies the length of the message.
- `flags` is not supported. Set `flags` to 0.
## Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMSGSIZE</td>
<td>The message is too long.</td>
</tr>
<tr>
<td>EHOSTUNREACH</td>
<td>No route to host.</td>
</tr>
<tr>
<td>ENOBUFS</td>
<td>Driver cannot allocate buffer.</td>
</tr>
<tr>
<td>ENOTCONN</td>
<td>Packets are sent through a non-established socket. The socket must be connected.</td>
</tr>
<tr>
<td>EWOULDBLOCK</td>
<td>The socket is marked non-blocking and the requested operation would block.</td>
</tr>
</tbody>
</table>
sendto()
Sends Message to Socket

Syntax
#include <sys/socket.h>
ssize_t sendto(
    int                 s,
    const void          *msg,
    size_t              len,
    int                 flags,
    struct sockaddr     *to,
    sockle_t            tolen);

Libraries
socket.l

Description
sendto() transmits a message to another socket. You may use the function with
unconnected sockets.

For reliable protocols, such as TCP, if no message space is available at the socket to
hold the message to transmit, send() normally blocks, unless the socket has been
placed in non-blocking I/O mode.

No indication of failure to deliver is implicit in a send. Return values of -1 indicate
some locally detected errors. On success, sendto() returns the number of characters
sent. On error, -1 is returned and errno is set to the error value.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
s
    specifies the path number of a socket created with socket().
buf
    points to the message to send.
len
    specifies the length of the message.
flags
    is not supported. Set flags to 0.

to
    points to the address of the target.

tolen
    specifies the size of the buffer associated with to.

Errors

EDSTADDRREQ  The to address pointer must be non-null and tolen variable
            must be non-zero.

EMSGSIZE     The message is too long.

EHOSTUNREACH No route to host.

ENOBUFFS     Driver cannot allocate buffer.

ENOTCONN     Packets are sent through a non-established socket.

EWOULDBLOCK  The socket is marked non-blocking and the requested
            operation would block.
sendmsg()

Send Message to Socket

Syntax

```c
#include <sys/socket.h>
ssize_t sendmsg(int s,
                  const struct msghdr *msg,
                  int flags);
```

Library

socket.l

Description

sendmsg() transmits a message to another socket. This function may be used with unconnected sockets.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe

Parameters

- `s` specifies the path number of a socket created with `socket()`.
- `msg` points to the message structure to send.
- `flags` specify `MSG_OOB`

Errors

- EMSGSIZE The message is too long.
- EHOSTUNREACH No route to host.
- ENOBUFFS Driver cannot allocate buffer.
- ENOTCONN Packets are sent through non-established socket.
- EWOULDBLOCK The socket is marked non-blocking and the requested operation would block.
sethostname()
Set Name of Current Host

Syntax

```
#include <sys/socket.h>
int sethostname(
    char    *name,
    size_t  namelen)
```

Libraries
socket.l

Description

`sethostname()` sets the host name for the current host.

If successful, `sethostname()` returns a value of 0. Otherwise, it returns -1 and places the appropriate error code in the global variable `errno`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `name` is a pointer to a null-terminated string containing the new standard host name for the current processor.

- `namelen` specifies the size of the name array.

Errors

- `EOS_MNF` inetdb module could not be found.

See Also

- `endnetent()`
- `getnetbyaddr()`
- `getnetbyname()`
- `getnetent()`
setsockopt()
Set Options on Sockets

Syntax

```c
#include <sys/socket.h>
int setsockopt(
    int s,
    int level,
    int optname,
    void *optval,
    socklen_t optlen)
```

Libraries

socket.l

Description

`setsockopt()` sets options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

`setsockopt()` returns 0 if the call succeeds. Otherwise, it returns -1 and sets `errno` to the error value.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Options

The include file `sys/socket.h` contains definitions for socket level options. Options at other protocol levels vary in format and name and are defined in the protocol's header file.

<table>
<thead>
<tr>
<th>Level</th>
<th>optname</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPPROTO_IP</td>
<td>IP_ADD_MEMBERSHIP</td>
<td>Join multicast group</td>
<td>ip_mreq()</td>
</tr>
<tr>
<td>IPPROTO_IP</td>
<td>IP_DROP_MEMBERSHIP</td>
<td>Leave multicast group</td>
<td>ip_mreq()</td>
</tr>
<tr>
<td>IPPROTO_IP</td>
<td>IP_MULTICAST_TTL</td>
<td>Set TTL for multicast packets</td>
<td>u_char</td>
</tr>
<tr>
<td>IPPROTO_IP</td>
<td>IP_MULTICAST_LOOP</td>
<td>Send multicast packets to the loopback interface</td>
<td>u_char</td>
</tr>
</tbody>
</table>
### Table 2-7. Options (Continued)

<table>
<thead>
<tr>
<th>Level</th>
<th>optname</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPPROTO_IP</td>
<td>IP_MULTICAST_IF</td>
<td>Select interface for sending multicast packets</td>
<td>in_addr{}</td>
</tr>
<tr>
<td>IPPROTO_TCP</td>
<td>TCP_MAXSEG</td>
<td>Set maximum segment size</td>
<td>int</td>
</tr>
<tr>
<td>IPPROTO_TCP</td>
<td>TCP_NODELAY</td>
<td>Do not delay send to coalesce packets</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_KEEPALIVE</td>
<td>Keep connection alive by forcing peer to respond periodically</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_LINGER</td>
<td>Linger on close if data present</td>
<td>linger{}</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_REUSEADDR</td>
<td>Allow local address reuse</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_BROADCAST</td>
<td>Permit sending broadcast datagrams</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_OOBINLINE</td>
<td>Leave out-of-band queue</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_SNDBUF</td>
<td>Set size of send buffer</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_RCVBUF</td>
<td>Set size of receive buffer</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_SNDDOWNAT</td>
<td>Minimum space required in send buffer to accept more data</td>
<td>int</td>
</tr>
<tr>
<td>SOL_SOCKET</td>
<td>SO_USELOOPBACK</td>
<td>Routing socket receives copy of any data sent (AF_ROUTE only)</td>
<td>int</td>
</tr>
</tbody>
</table>

### Parameters

- **s**
  
  specifies the path number of the socket.

- **level**
  
  specifies where the option resides.

  - To set options at the socket level, specify `level` as `SOL_SOCKET`.
  
  - To set options at any other level, supply the protocol number of the appropriate protocol controlling the option. For example, `IPPROTO_TCP` specifies TCP options.
optname
    is the name of the option to be set.

optval
    is the new value of the option.

optlen
    is the size of optval in bytes.

Errors

ENOPROTOOPT
    The option is unknown.

EFAULT
    The address pointed to by optval is not in a valid part of the process address space.

See Also

getsockopt()
socket()
**shutdown()**

Shut Down Part of Full-Duplex Connection

**Syntax**

```c
#include <sys/socket.h>
int shutdown(
    int s,
    int how)
```

**Libraries**

socket.l

**Description**

`shutdown()` shuts down all or part of a full-duplex connection of the socket specified by `s`.

If successful, `shutdown()` returns a value of 0. Otherwise, it returns -1 and places the appropriate error code in the global variable `errno`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `s` specifies the path number of the socket.
- `how` specifies the method for receiving and sending permissions.
  - If `how` is 0, further receives are disallowed.
  - If `how` is 1, further sends are disallowed.
  - If `how` is 2, further sends and receives are disallowed.

**Errors**

- ENOTCONN The specified socket is not connected.

**See Also**

`connect()`  
`socket()`
socket()

Creates Endpoint for Communication

Syntax

```c
#include <sys/types.h>
#include <sys/socket.h>
int socket(
    int     af,
    int     type,
    int     protocol)
```

Libraries
socket.l

Description

`socket()` creates an endpoint for communication and returns a path number.
If successful, `socket()` returns the descriptor referencing the socket. Otherwise, it returns -1 and places the appropriate error code in the global variable `errno`.

SOCK_STREAM Sockets

Sockets of type SOCK_STREAM are sequenced, reliable, and full-duplex byte streams, similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A `connect()` call creates a connection to another socket.
Once connected, data may be transferred using `_os_read()` and `_os_write()` calls or some variant of the `send()` and `recv()` calls.

SOCK_DGRAM Sockets

SOCK_DGRAM sockets provide an unreliable datagram service. The socket may be either connected or unconnected. If it is connected, data may be sent and received using `read()`, `write()`, `_os_read()`, `_os_write()`, `send()` or `recv()`. If the socket is unconnected, `sendto()`, `sendmsg()`, `recvfrom()`, and `recvmsg()` must be used.

SOCK_RAW Sockets

SOCK_RAW allows you to build datagrams, including headers. It is used to send and receive ICMP messages, internal routing requests, and user-defined protocols. Sockets of this type require super-user privileges to create.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Parameters

**af**

specifies an address format with which addresses specified in later operations using the socket should be interpreted.

These formats are defined in the include file `socket.h`. Accepted formats are listed below:

- **AF_INET**: ARPA Internet address family.
- **AF_INET6**: IP version 6
- **AF_ROUTE**: Internal Routing Protocol

**type**

specifies the semantics of communication.

Type values are listed below:

- **SOCK_STREAM**: provides sequenced, reliable, two-way connection based byte streams with an out-of-band data transmission mechanism.
- **SOCK_DGRAM**: supports datagrams (connectionless, unreliable messages of a fixed and small maximum length).
- **SOCK_RAW**: supports datagrams (connectionless, unreliable messages of a fixed and typically small maximum length).

**protocol**

specifies a particular protocol to use with the socket.

Normally only a single protocol exists to support a particular socket type, using a given address format. However, many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number is particular to the communication domain in which communication is to take place.

Errors

**EAFNOSUPPORT**

The specified address family is not supported in this version of the system.

**EPROTONOSUPPORT**

Specified protocol is not supported.

**ENOBUFS**

There is no end-buffer space available. The socket cannot be created.

See Also

`accept()` `bind()` `getsockname()` `getsockopt()` `listen()` `recv()` `send()` `shutdown()`
Other OS-9 Functions

The following generic OS-9 system calls are also supported by sockets.

- _os_close()
- _os_gs_popt()
- _os_gs_ready()
- _os_read()
- _os_ss_popt()
- _os_ss_relea()
- _os_ss_sendsig()
- _os_write()
- ioctl()
- select()
Point-to-Point Protocol Functions

The Point-to-Point Protocol (PPP) Application Programming Interface (API) provides four types of function calls: stack configuration, CHAT scripting, authentication database, and connection/disconnect. In addition, this API defines structures that provide error reporting and other functionalities between the PPP stack and the software using the API.

The PPP API functions are listed in the `ppplib.l` library. The following table lists and describes the API functions that compose the `ppplib.l` file:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ppp_auth_add_chap()</code></td>
<td>Add CHAP Entry to Authentication Module</td>
</tr>
<tr>
<td><code>ppp_auth_add_pap()</code></td>
<td>Add PAP Entry to Authentication Module</td>
</tr>
<tr>
<td><code>ppp_auth_create_mod()</code></td>
<td>Create a New Authentication Module (Database)</td>
</tr>
<tr>
<td><code>ppp_auth_del_chap()</code></td>
<td>Delete CHAP Entry from Authentication Module</td>
</tr>
<tr>
<td><code>ppp_auth_del_pap()</code></td>
<td>Delete PAP Entry from Authentication Module</td>
</tr>
<tr>
<td><code>ppp_auth_get_cur_chap()</code></td>
<td>Get CHAP Name/Secret for Currently Set Peer</td>
</tr>
<tr>
<td><code>ppp_auth_get_cur_pap()</code></td>
<td>Get PAP Name/Secret for Currently Set Peer</td>
</tr>
<tr>
<td><code>ppp_auth_get_peer_name()</code></td>
<td>Get Name of Currently Set Peer</td>
</tr>
<tr>
<td><code>ppp_auth_link_mod()</code></td>
<td>Link to Existing Authentication Module</td>
</tr>
<tr>
<td><code>ppp_auth_set_peer_name()</code></td>
<td>Set Current Peer</td>
</tr>
<tr>
<td><code>ppp_auth_unlink_mod()</code></td>
<td>Unlink from an Authentication Module</td>
</tr>
<tr>
<td><code>ppp_chat_close()</code></td>
<td>Close CHAT Path</td>
</tr>
<tr>
<td><code>ppp_chat_open()</code></td>
<td>Open a Raw CHAT Path</td>
</tr>
<tr>
<td><code>ppp_chat_read()</code></td>
<td>Read Data from CHAT Path</td>
</tr>
<tr>
<td><code>ppp_chat_script()</code></td>
<td>Run a CHAT Script</td>
</tr>
<tr>
<td><code>ppp_chat_write()</code></td>
<td>Write Data to CHAT Path</td>
</tr>
<tr>
<td><code>ppp_close()</code></td>
<td>Close the PPP Stack</td>
</tr>
<tr>
<td><code>ppp_connect()</code></td>
<td>Run Optional CHAT Script and Establish PPP Link</td>
</tr>
<tr>
<td><code>ppp_disconnect()</code></td>
<td>Terminate Current PPP Link</td>
</tr>
<tr>
<td><code>ppp_get_asynch_params()</code></td>
<td>Get Asynchronous Link Parameters</td>
</tr>
<tr>
<td><code>ppp_get_options()</code></td>
<td>Get Negotiable Stack Options</td>
</tr>
<tr>
<td><code>ppp_get_params()</code></td>
<td>Obtain Negotiated Stack Parameters</td>
</tr>
<tr>
<td><code>ppp_get_statistics()</code></td>
<td>Obtain Current Stack Statistics</td>
</tr>
<tr>
<td><code>ppp_init()</code></td>
<td>Initialize the PPP API</td>
</tr>
<tr>
<td><code>ppp_open()</code></td>
<td>Open the PPP Stack</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>ppp_reset_statistics()</td>
<td>Reset Stack Statistics</td>
</tr>
<tr>
<td>ppp_set_asynch_params()</td>
<td>Set Asynchronous Link Parameters</td>
</tr>
<tr>
<td>ppp_set_options()</td>
<td>Set Negotiable Stack Options</td>
</tr>
<tr>
<td>ppp_start()</td>
<td>Establish a PPP Link</td>
</tr>
<tr>
<td>ppp_term()</td>
<td>Terminate the PPP API</td>
</tr>
</tbody>
</table>
**ppp_auth_add_chap()**  
Add CHAP Entry to Authentication Module

**Syntax**
```c
#include <SPF/ppplib.h>
error_code
ppp_auth_add_chap(
    char        *peer_name,
    char        *id,
    char        *secret,
    auth_handle *hndl);
```

**Libraries**
ppplib.l

**Description**
This function adds new CHAP peer/ID/secret group to the authentication module. The peer name must be unique; each peer may only have only one CHAP entry and one PAP entry. If a peer already has a CHAP entry within the database, the existing entry is overwritten.

**Attributes**
Operating System: OS-9  
State: User  
Threads: Safe

**Parameters**
- `peer_name` is a pointer to new peer name.  
  This string should not exceed PPP_MAX_PEER_NAME in length.
- `id` is a pointer to new name.  
  This string should not exceed PPP_MAX_NAME bytes in length.
- `secret` is a pointer to new CHAP secret.  
  This string should not exceed PPP_MAX_SECRET bytes in length.
- `hndl` is a handle to the authentication module.  
  This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.
### Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_NOTRDY</td>
<td>Returned when the API has not been initialized.</td>
</tr>
<tr>
<td>EOS_FULL</td>
<td>No more free entries within the authentication module.</td>
</tr>
<tr>
<td>EOS_ILLPRM</td>
<td>Illegal parameter.</td>
</tr>
</tbody>
</table>
### Syntax

```
#include <SPF/ppplib.h>
error_code ppp_auth_add_pap(
    char        *peer_name,
    char        *id,
    char        *secret,
    auth_handle *hndl);
```

### Libraries

ppplib.l

### Description

This function adds new PAP peer/ID/secret group to the authentication module. The peer name must be unique in that each peer may only have one CHAP entry and one PAP entry. If a peer already has a PAP entry within the database, the existing entry is overwritten.

### Attributes

**Operating System:** OS-9  
**State:** User  
**Threads:** Safe

### Parameters

- **peer_name**  
  is a pointer to new peer name.  
  This string should not exceed PPP_MAX_PEER_NAME in length.

- **id**  
  is a pointer to new name.  
  This string should not exceed PPP_MAX_NAME bytes in length.

- **secret**  
  is a pointer to new PAP secret.  
  This string should not exceed PPP_MAX_SECRET bytes in length.

- **hndl**  
  is a handle to the authentication module.  
  This parameter is obtained from ppp_auth_create_mod() or ppp_auth_link_mod().
## Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_NOTRDY</td>
<td>Returned when the API has not been initialized.</td>
</tr>
<tr>
<td>EOS_FULL</td>
<td>Allows no more free entries within the authentication module.</td>
</tr>
<tr>
<td>EOS_ILLPRM</td>
<td>Illegal parameter.</td>
</tr>
</tbody>
</table>
**ppp_auth_create_mod()**

Create a New Authentication Module (Database)

**Syntax**

```c
#include <SPF/ppplib.h>

error_code
ppp_auth_create_mod(
    u_int16 max_entries,
    auth_handle *hndl);
```

**Libraries**

ppplib.l

**Description**

This function creates a new authentication module. This module is used to store authentication information when PAP or CHAP authentication is being used to connect to a remote peer.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `max_entries` is the maximum of host/ID/secret groups to be stored in the module.
- `hndl` is the location where `ppp_auth_create_mod()` will store a handle to the new authentication module. This handle will be used in all subsequent calls to the authentication functions in this library.

**Errors**

- EOS_KWNMOD signifies that the authentication module already exists.

  If this error appears, a call to `ppp_auth_link_mod()` should be made.

**See Also**

- `auth_handle`
- `ppp_auth_link_mod()`
**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_auth_del_chap(
    char          *peer_name,
    auth_handle   *hndl);
```

**Libraries**

ppplib.l

**Description**

This function deletes existing CHAP entry for specified peer from the authentication module.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- **peer_name** is a pointer to peer name.
  
  This string should not exceed PPP_MAX_PEER_NAME in length.

- **hndl** is a handle to the authentication module.
  
  This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

**Errors**

- **EOS_NOTRDY** Returned when the API has not been initialized.
- **EOS_PNNF** No CHAP entry for specified peer.
**ppp_auth_del_pap()**

Delete PAP Entry from Authentication Module

**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_auth_del_pap(
    char          *peer_name,
    auth_handle   *hndl);
```

**Libraries**

ppplib.l

**Description**

This function deletes existing PAP entry for specified peer from the authentication module.

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Threads:** Safe

**Parameters**

- **peer_name**
  - is a pointer to peer name.
  - This string should not exceed PPP_MAX_PEER_NAME in length.

- **hndlis** a handle to the authentication module.
  - This parameter is obtained from ppp_auth_create_mod() or ppp_auth_link_mod().

**Errors**

- **EOS_NOTRDY**
  - Returned when the API has not been initialized.
- **EOS_PNNF**
  - No PAP entry for specified peer.
**ppp_auth_get_cur_chap()**

Get CHAP Name/Secret for Currently Set Peer

**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_auth_get_cur_chap(
    char           *name,
    char           *secret,
    auth_handle    *hndl);
```

**Libraries**

ppplib.l

**Description**

This call gets the CHAP secret needed to connect to the current host.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `name` is the location of a character array containing the CHAP name. This NULL-terminated character array should no more than `PPP_MAX_NAME` bytes in length.

- `secret` is the location of a character array where the CHAP secret will be placed. This character array should be at most `PPP_MAX_SECRET` bytes in length.

- `hndl` is the handle to the authentication module. This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

**Errors**

- `EOS_NOTRDY` Returned when the API has not been initialized.
- `EOS_BNAM` Signifies that the secret could not be found for the current host and name.

**See Also**

- `ppp_auth_get_peer_name()` `ppp_auth_set_peer_name()`
ppp_auth_get_cur_pap()

Get PAP Name/Secret for Currently Set Peer

Syntax
#include <SPF/ppplib.h>
error_code
ppp_auth_get_cur_pap(
    char          *name,
    char          *secret,
    auth_handle   *hndl);

Libraries
ppplib.l

Description
This call gets the PAP secret needed to connect to the current host.

Attributes
Operating System:  OS-9
State:            User
Threads:          Safe

Parameters
name
is the location of a character array containing the PAP name.
This character array must be no more than PPP_MAX_NAME bytes in length.

secret
is the location of a character array where the PAP secret will be placed.
This character array should be at least PPP_MAX_SECRET bytes in length.

hndl
is the handle to the authentication module.
This parameter is obtained from ppp_auth_create_mod() or
ppp_auth_link_mod().

Errors
EOS_NOTRDY     Returned when the API has not been initialized.
EOS_BNAM       Secret could not be found for the current host and name.

See Also
ppp_auth_get_peer_name() ppp_auth_set_peer_name()
ppp_auth_get_peer_name()
Get Name of Currently Set Peer

Syntax
#include <SPF/ppplib.h>
error_code
ppp_auth_get_peer_name(
    char           *name,
    auth_handle    *hndl);

Libraries
ppplib.l

Description
This call gets the current remote peer name from the authentication module, as was set with a previous ppp_auth_set_peer_name() call.

Attributes
Operating System: OS-9
State: User
Threads: Safe

Parameters
name
    is the location to store the current peer name.
    This buffer should be at least PPP_MAX_PEER_NAME bytes in length.

hndl
    is the handle to the authentication module.
    This parameter is obtained from ppp_auth_create_mod() or ppp_auth_link_mod().

Errors
EOS_NOTRDY     Returned when the API has not been initialized.

See Also
ppp_auth_set_peer_name()
**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_auth_link_mod(auth_handle *hndl);
```

**Libraries**

`ppplib.l`

**Description**

This function links to an existing authentication module. It is used to store authentication information when PAP or CHAP authentication is being used to connect to a remote peer.

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Threads:** Safe

**Parameters**

`hndl`

- is the location where `ppp_auth_create_mod()` will store a handle to the new authentication module.

This handle will be used in all subsequent calls to the authentication functions in this library.

**Errors**

- **EOS_NOTRDY**
  - Returned when the API has not been initialized.

- **EOS_MNF**
  - Signifies that the authentication module was not found in memory.
  - If this error appears, a call to `ppp_auth_create_mod()` should be made.

**See Also**

- `auth_handle`
- `ppp_auth_create_mod()`
**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_auth_set_peer_name(
    char          *name,
    auth_handle   *hndl);
```

**Libraries**

ppplib.l

**Description**

This function sets the current remote peer name in the authentication module. This determines which values (such as PAP/CHAP names and secrets) will be get/set in subsequent authentication calls.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `name` is the location of the new remote peer name.
  This character array should be no more than PPP_MAX_PEER_NAME bytes in length.

- `hndl` is the handle to the authentication module.
  This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

**Errors**

- EOS_NOTRDY Returned when the API has not been initialized.

**See Also**

- `ppp_auth_get_peer_name()`
**ppp_auth_unlink_mod()**

Unlink from an Authentication Module

**Syntax**

```
#include <SPF/ppplib.h>
error_code
ppp_auth_unlink_mod(auth_handle *hndl);
```

**Libraries**

ppplib.l

**Description**

This call unlinks from an authentication module previously linked to with `ppp_auth_link_mod()`, or created with `ppp_auth_create_mod()`.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- *hndl*
  - handle of the authentication module to unlink.

**Errors**

- **EOS_NOTRDY**
  - Returned when API has not been initialized.
- **EOS_MNF**
  - Signifies that the authentication module was not found in memory.

**See Also**

- `auth_handle`
- `ppp_auth_create_mod()`
- `ppp_auth_link_mod()`
ppp_chat_close()

Close CHAT Path

Syntax

```c
#include <SPF/ppplib.h>
error_code
ppp_chat_close(path_id chat_path);
```

Libraries

ppplib.l

Description

This function closes the path opened by `ppp_chat_open()`.

Attributes

Operating System: OS-9
State: User
Threads: Safe

Parameters

chat_path
Path identifier that was returned from a successful call to `ppp_chat_open()`.

Errors

- EOS_NOTRDY: Returned when the API has not been initialized.
- EOS_BPNUM: Returned when this not a valid path identifier.

See Also

- `ppp_chat_read()`
- `ppp_chat_write()`
- `ppp_open()`
- `ppp_start()`
ppp_chat_open()

Open a Raw CHAT Path

Syntax

```c
#include <SPF/ppplib.h>
error_code
ppp_chat_open(
    char       *hdlc_name,
    path_id    *chat_path);
```

Libraries

ppplib.l

Description

This function opens a path to the HDLC driver and places the driver into CHAT mode. When in this mode, the caller can use the `chat_path` to read/write directly from or to the communications device below the HDLC layer.

The name of the HDLC descriptor should be the same as the one used within the PPP stack name. For example, the PPP stack name of `</dev>/hdlc0/lcp0/ipcp0` would use `/hdlc0` for the `hdlc_name` field.

Attributes

Operating System: OS-9
State: User
Threads: Safe

Parameters

- `stack_name` is the name of the HDLC descriptor contained within the PPP stack name.
- `chat_path` returns the path of the HDLC layer at this location.

Errors

- EOS_NOTRDY Returned when the API has not been initialized.
- EOS_BPNUM Returned when this not a valid PPP stack path identifier.

See Also

- `ppp_chat_close()`, `ppp_chat_read()`, `ppp_chat_write()`, `ppp_open()`, `ppp_start()`
ppp_chat_read()
Read Data from CHAT Path

Syntax
#include <SPF/ppplib.h>
error_code
ppp_chat_read(
    path_id m   chat_path,
    void        *buffer,
    u_int32      *count);

Libraries
ppplib.l

Description
This function reads user data from the path. Use this call after ppp_chat_open() to read data from the data port without HDLC decoding.

Attributes
Operating System: OS-9
State: User
Threads: Safe

Parameters
chat_path is the path identifier that was returned from a successful call to ppp_chat_open().
buffer is the pointer to the caller's character buffer.
count is the pointer to the number of bytes in the caller’s character buffer.

After the call completes, the number of bytes actually read is returned here.

Errors
EOS_NOTRDY Returned when the API has not been initialized.
EOS_BPNUM Returned when this not a valid PPP stack path identifier.

See Also
ite_data_read()
_os_read()
ppp_chat_write()
**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_chat_script(
    path_id        chat_path,
    path_id        log_path,
    u_int8         chat_type,
    char           *chat_name,
    ppp_conninfo   *ci);
```

**Libraries**

ppplib.l

**Description**

This function runs a CHAT script from either a text file or a data module. An optional logging path may be specified where CHAT commands and responses will be echoed. A log path of `PPP_NOLOG` denotes no logging is desired.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `chat_path` is the CHAT path identifier that was returned from a successful call to `ppp_chat_open()`.
- `log_path` is the path where CHAT commands and responses will be echoed for logging purposes. The `log_path` of `PPP_NOLOG` may be used to prevent logging.
- `chat_type` specifies the container of the CHAT script commands: `PPP_CHAT_TYPE_MODULE` or `PPP_CHAT_TYPE_FILE`. 
chat_name
    is the name of the CHAT module or file.

ci
    is a pointer to structure allocated by the application that contains connection
    information for the life of the connection.

    Applications must keep this memory structure to prevent errors from
    resulting.

**Errors**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_NOTRDY</td>
<td>Returned when the API has not been initialized.</td>
</tr>
<tr>
<td>EOS_BPNUM</td>
<td>CHAT or log path is invalid.</td>
</tr>
<tr>
<td>EOS_ILLFNC</td>
<td>Unknown command in CHAT script.</td>
</tr>
<tr>
<td>EOS_ILLPRM</td>
<td>Bad argument in CHAT script.</td>
</tr>
<tr>
<td>EOS_NORAM</td>
<td>Unable to allocate CHAT engine memory.</td>
</tr>
<tr>
<td>EOS_PPP_CHAT_BADSTR</td>
<td>Malformed string in CHAT script.</td>
</tr>
<tr>
<td>EOS_PPP_CHAT_ABORT</td>
<td>Script aborted due to reception of an ABORT string.</td>
</tr>
<tr>
<td>EOS_PPP_CHAT_APPABORT</td>
<td>Script aborted due to application setting the PPP_CIFLAG_CHATABORT flag within the ppp_conninfo data structure.</td>
</tr>
<tr>
<td>ETIMEDOUT</td>
<td>Communication with the remote peer timed out.</td>
</tr>
</tbody>
</table>

**See Also**

- `ppp_chat_close()`
- `ppp_chat_open()`
- `ppp_start()`
ppp_chat_write()

Write Data to CHAT Path

Syntax

```c
#include <SPF/ppplib.h>
error_code
ppp_chat_write(
    path_id    chat_path,
    void       *buffer,
    u_int32    *count);
```

Libraries

ppplib.1

Description

This function writes user data down the CHAT path. Use this function after `ppp_chat_open()` to send data out the data port without HDLC framing.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe

Parameters

- `chat_path` is a path identifier returned from a successful call to `ppp_chat_open()`.
- `buffer` is a pointer to the caller's character buffer.
- `count` is a pointer to the number of bytes in the caller's character buffer. After the call is completed, the number of bytes actually written is returned here.

Errors

- `EOS_NOTRDY` Returned when the API has not been initialized.
- `EOS_BPNUM` Returned when this not a valid PPP stack path identifier.

See Also

- `_os_write()`
- `ppp_chat_read()`
- `ppp_chat_read()`
**ppp_close()**

Close the PPP Stack

**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_close(path_id stack_path);
```

**Libraries**

ppplib.l

**Description**

This function closes the PPP stack associated with the `stack_path` handle.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

`stack_path` is a path identifier returned from a successful call to `ppp_open()`.

**Errors**

- **EOS_NOTRDY**: Returned when API has not been initialized.
- **EOS_DEVBSY**: Returned if PPP link is still established.
- **EOS_BPNUM**: Returned when not a valid PPP stack path identifier.

**See Also**

`ppp_open()`
ppp_connect()

Run Optional CHAT Script and Establish PPP Link

Syntax

```c
#include <SPF/ppplib.h>
error_code
ppp_connect(
    path_id       stack_path,
    path_id       log_path,
    char          *hdlc_name,
    u_int8        chat_type,
    char          *chat_name,
    ppp_conninfo  *connection_info);
```

Libraries

ppplib.l

Description

This function connects an open PPP stack to a remote peer. It can be direct connect or dial-up. If the connection is dial-up, the function can parse a data module or disk file (a CHAT script) into send/expect command pairs, which are sent/received to/from a modem to establish the connection with the remote peer.

This function will return when the modems have connected and HDLC is enabled, when the protocol has timed out, or when the function encounters an error in the CHAT commands. If `stack_up_sig` and `stack_down_sig` in the `connection_info` structure are non-zero, the application will receive one or the other when the connection attempt resolves. If the signals are zero, the calling process will be sent to no signals.

If it is necessary to run a customized chat script, use the `chat_open()`, `chat_write()`, `chat_read()`, `chat_close()` and `ppp_start()` calls instead of the `ppp_connect()` call.

Attributes

Operating System:   OS-9
State:              User
Threads:            Safe
Parameters

stack_path
is a path identifier returned from a successful call to ppp_open().

log_path
is a path where CHAT commands and responses will be echoed for logging purposes. log_path of PPP_NOLOG may be used for this parameter to prevent logging.

hdlc_name
is the name of the HDLC descriptor contained within the PPP stack name.
For example, the PPP stack name of </dev>/hdlc0/lcp0/ipcp0 would use /hdlc0 for this parameter.

chat_type
specifies the container of the send/expect commands: PPP_CHAT_TYPE_MODULE, PPP_CHAT_TYPE_FILE, or PPP_CHAT_TYPE_NONE (if no CHAT script is required).

chat_name
is the name of the chat module/file (or NULL pointer if chat_type is PPP_CHAT_TYPE_NONE).

connection_info
is a pointer to structure allocated by the application containing connection information for the life of the connection.
Applications must keep this memory structure to prevent errors from resulting.

Errors

EOS_NOTRDY Returned when the API has not been initialized.
EOS_DEVBSY Returned when a PPP link is established.
EOS_BPNUM Returned when this not a valid PPP stack path identifier.
EOS_PARAM There was an error in the chat commands.
EOS_MNF Module not found.
EOS_PNNF Disk file not found.
ETIMEDOUT Communication with the remote peer timed out.

See Also

_os_intercept()
ppp_chat_open()
ppp_chat_write()
ppp_get_params()
ppp_open()
ppp_start()
ppp_disconnect()

Terminate Current PPP Link

Syntax

```c
#include <SPF/ppplib.h>
error_code
ppp_disconnect(path_id stack_path);
```

Libraries

ppplib.l

Description

This call specifies a disconnect from the remote peer. This is done by sending a
terminate request message to the remote peer. Also, a request is made to drop the
modem carrier to the driver below the HDLC layer. After this call, it is safe for the
application to return the memory for the `ppp_conninfo` structure passed in from the
`ppp_connect()` or `ppp_start()`.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe

Parameters

- `stack_path` is a path identifier returned from a successful call to `ppp_open()`.

Errors

- `EOS_NOTRDY` Returned when the API has not been initialized or if a PPP
  link is not established.
- `EOS_BPNUM` Returned when this not a valid PPP path identifier.

See Also

- `ppp_connect()`
- `ppp_open()`
**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_get_asynch_params(
    path_id       stack_path,
    ppp_modem_p   *params);
```

**Libraries**

`ppplib.l`

**Description**

This call gets the parameter values for the asynchronous PPP link. The desired options are returned in the `ppp_modem_p` structure.

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Threads:** Safe

**Parameters**

- `stack_path` is a path identifier that was returned from a successful call to `ppp_open()`.
- `params` is a pointer to the location where `ppp_get_asynch_params()` will store the current asynchronous parameter values for the PPP link.

**Errors**

- `EOS_NOTRDY` Returned when the API has not been initialized.
- `EOS_BPNUM` Returned when this not a valid PPP stack path identifier.
- `EOS_PERMIT` Invalid pointer.
- `EOS_ILLPRM` Illegal parameter.

**See Also**

- `ppp_connect()`, `ppp_modem_p`  
- `ppp_open()`, `ppp_set_asynch_params()`  

*(`ppp_set_asynch_params()` is valid only for asynchronous PPP links.)*
ppp_get_options()

Get Negotiable Stack Options

Syntax
#include <SPF/ppplib.h>
error_code
ppp_get_options(
  path_id            stack_path,
  ppp_option_block   *options);

Libraries
ppplib.l

Description
This call gets the negotiable options of the drivers in an open PPP stack. The options
will be stored in the specified ppp_option_block.

Attributes
Operating System:       OS-9
State:                  User
Threads:                Safe

Parameters
stack_path
  is a path identifier that was returned from a successful call to ppp_open().

options
  is a pointer to the location at which ppp_get_options() will store the current
  options for the PPP stack.

Errors
EOS_NOTRDY            Returned when the API has not been initialized.
EOS_BPNUM             Returned when this not a valid PPP stack path identifier.
EOS_PERMIT            Invalid pointer.
EOS_ILLPRM            Illegal parameter.

See Also
ppp_connect()
ppp_open()
ppp_option_block
ppp_set_options()
ppp_get_params()
Obtain Negotiated Stack Parameters

Syntax
#include <SPF/ppplib.h>
error_code
ppp_get_params(
    path_id          stack_path,
    ppp_param_block *params);

Libraries
ppplib.1

Description
This call obtains the current negotiated link parameters of the drivers in an open
PPP stack. This information will be placed at the location of the indicated
ppp_param_block structure.

Attributes
Operating System: OS-9
State: User
Threads: Safe

Parameters
stack_path
    is a path identifier that was returned from a successful call to ppp_open().

params
    is a pointer to the location where ppp_get_params() will store the current
    negotiated parameters of the PPP stack.

Errors
EOS_NOTRDY    Returned when API has not been initialized.
EOS_BPNUM     Returned when this not a valid PPP stack path identifier.

See Also
ppp_get_options()
ppp_open()
ppp_param_block
ppp_set_options()
ppp_get_statistics()

Obtain Current Stack Statistics

Syntax

#include <SPF/ppplib.h>

error_code

ppp_get_statistics(
  path_id          stack_path,
  ppp_ipcp_stats   *ipcp_stats,
  ppp_lcp_stats    *lcp_stats,
  ppp_hdlc_stats   *hdlc_stats);

Libraries

ppplib.l

Description

This function queries the drivers in the specified PPP stack and returns current statistics for each layer. A NULL pointer may be passed in for any layer in which the caller is not interested.

The LCP and IPCP layers do not currently support statistics and therefore must be NULL.

Attributes

Operating System: OS-9
State: User
Threads: Safe

Parameters

stack_path
  is a path identifier that was returned from a successful call to ppp_open().

ipcp_stats
  is a pointer to a user-allocated statistics structure where
  ppp_get_statistics() will return the IPCP-layer statistics.

lcp_stats
  is a pointer to a user-allocated statistics structure where
  ppp_get_statistics() will return the LCP-layer statistics.

hdlc_stats
  is a pointer to a user-allocated statistics structure where
  ppp_get_statistics() will return the HDLC-layer statistics.
Errors

- **EOS_NOTRDY**: Returned when the API has not been initialized.
- **EOS_PERMIT**: Invalid pointer was passed in.
- **EOS_BPNUM**: Returned when this not a valid PPP stack path identifier.

See Also

- `ppp_open()`
**ppp_init()**

Initialize the PPP API

**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_init(void     *rsvd);
```

**Libraries**

ppplib.l

**Description**

This call initializes the PPP API. This call must be made before any other calls are allowed. Currently, the single `rsvd` parameter is reserved for future use and must be NULL.

**Attributes**

Operating System: OS-9

State: User

Threads: Safe

**Parameters**

`rsvd`

Reserved for possible future use; must be set to NULL.

**Errors**

`EOS_DEVBSY` Returned when the API has been initialized.

**See Also**

`ppp_close()`

`ppp_disconnect()`
**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_open(
    char *stack_name,
    path_id *stack_path);
```

**Libraries**

ppplib.l

**Description**

This is the PPP function that opens the PPP stack.

This function merely opens a path to the stack. To actually connect to a remote peer use `ppp_connect()` or `ppp_start()` after the `ppp_open()`. The path identifier that is returned from a successful `ppp_open()` should eventually be used in a call to `ppp_close()`.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `stack_name` is a NULL-terminated string containing the name of the PPP stack (`/hdlc0/lcp0/ipcp0`).
- `stack_path` is a pointer to the location where `ppp_open()` will store the path identifier of the opened path.

**Errors**

- EOS_NOTRDY Returned when the API has not been initialized.
- EOS_MNF Returned when the specified descriptors are not found in memory.

**See Also**

- `ppp_close()`
- `ppp_disconnect()`
**Syntax**

```
#include <SPF/ppplib.h>

error_code ppp_reset_statistics(
    path_id stack_path,
    u_int16 layers);
```

**Libraries**

ppplib.l

**Description**

This function resets the statistics in the specified PPP stack. `layers` is a bitmask that determines which layers should reset their statistics. Currently, only the HDLC layer supports statistics; therefore, the only valid value for `layers` is `PPP_LAYER_HDLC`.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `stack_path` is a path identifier that was returned from a successful call to `ppp_open()`.
- `layers` is a bitmask that determines which layers should reset their statistics. Layer bit values are defined in `SPF/ppplib.h` (`PPP_LAYER_IPCP`, `PPP_LAYER_LCP`, and `PPP_LAYER_HDLC`).

**Errors**

- `EOS_NOTRDY`: Returned when the API has not been initialized.
- `EOS_BPNUM`: Returned when this not a valid PPP stack path identifier.

**See Also**

`ppp_open()`
**ppp_set_asynch_params()**

*Set Asynchronous Link Parameters*

**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_set_asynch_params(
    path_id        stack_path,
    ppp_modem_p    *params);
```

**Libraries**

ppplib.l

**Description**

This call sets the configurable parameter values for the asynchronous PPP link. The desired options are specified in the `ppp_modem_p` structure. This structure may be filled out completely by the user or selected items may be updated in a `ppp_modem_p` returned from a successful call to `ppp_get_asynch_params()`. The configuration must be set before the call to `ppp_connect()` or `ppp_start()` for the given path.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `stack_path` is a path identifier returned from a successful call to `ppp_open()`.
- `params` is a pointer to the location that `ppp_set_asynch_params()` uses to update the asynchronous parameter values for the PPP link.

**Errors**

- **EOS_NOTRDY** Returned when the API has not been initialized.
- **EOS_BPNUM** Returned when this not a valid PPP stack path identifier.
- **EOS_ILLPRM** Illegal value was specified for one of the options.
See Also

ppp_connect()

ppp_get_asynch_params()

ppp_modem_p

ppp_open()

(pppp_open() is valid only for asynchronous PPP links. The rx_dev_name and tx_dev_name fields of the PPP parameter structure should refer to the same device. The rx_dev_name field is used to specify the device that the PPP stack will use for communication. The tx_dev_name field is ignored.)
Set Negotiable Stack Options

Syntax

```c
#include <SPF/ppplib.h>

error_code ppp_set_options(
    path_id           stack_path,
    ppp_option_block  *options);
```

Libraries

ppplib.l

Description

This call sets the negotiable options of the drivers in an open PPP stack. The options will be taken from the specified `ppp_option_block`. This structure can be filled out completely by the user or selected items can be modified in a `ppp_option_block` returned from a successful call to `ppp_get_options()`. The configuration must be set before the call to `ppp_connect()` or `ppp_start()` for the given path.

Attributes

Operating System: OS-9
State: User
Threads: Safe

Parameters

- `stack_path` is a path identifier returned from a successful call to `ppp_open()`.
- `options` is a pointer to the location where `ppp_set_options()` will take the new options for the PPP stack.

Errors

- `EOS_NOTRDY` Returned when the API has not been initialized.
- `EOS_BPNUM` Returned when this not a valid PPP stack path identifier.
- `EOS_ILLPRM` Illegal value was specified for one of the options.

See Also

- `ppp_connect()`
- `ppp_get_options()`
- `ppp_open()`
- `ppp_option_block`
**ppp_start()**

Establish a PPP Link

**Syntax**

```c
#include <SPF/ppplib.h>
error_code
ppp_start(
    path_id        stack_path,
    ppp_conninfo   *connection_info);
```

**Libraries**

ppplib.l

**Description**

This function completes the PPP connection after the CHAT operation has been performed. This call does not use the `chat_path`, rather the `stack_path`, which contains the complete PPP signaling stack.

The `chat_path` must have been closed prior to making this call in order to allow the HDLC layer to begin HDLC framing.

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**

- `stack_path` is a path identifier that was returned from a successful call to `ppp_open()`.
- `connection_info` is a pointer to structure allocated by the application that contains connection information for the life of the connection.

Applications must keep this memory structure to prevent errors from resulting.

**Errors**

- **EOS_NOTRDY** Returned when the API has not been initialized.
- **EOS_BPNUM** Returned when this not a valid PPP stack path identifier.
- **EOS_PARAM** Error in the chat commands.
- **ETIMEDOUT** Communication with the remote peer timed out.

**See Also**

`ppp_open()`
**ppp_term()**

Terminate the PPP API

### Syntax

```c
#include <SPF/ppplib.h>

error_code
ppp_term(void *rsvd);
```

### Libraries

ppplib.l

### Description

This call terminates (deinitializes) the use of this API. Currently, the single `rsvd` parameter is reserved for future use and must be NULL.

### Attributes

- **Operating System:** OS-9
- **State:** User
- **Threads:** Safe

### Parameters

- `rsvd` is reserved for possible future use; must be set to NULL.

### Errors

- **EOS_NOTRDY** Returned when the API has not been initialized.

### See Also

- `ppp_close()`
- `ppp_connect()`
Per-Path Static Storage Functions

A library (ppstat.l) is supplied with the SoftStax package that facilitates usage of per-path storage (PPS). PPS allows a protocol driver to store unique information for each path that it is currently open on a particular logical unit. The PPS structures are simply stored in a linked list pointed to by the lu_pps_list field of the SPF logical unit static storage.

There are two sections of the PPS structure: the ppstat.l section and the driver’s section. ppstat.l’s section is defined in ppstat.h. The driver’s section is specified by defining the pre-processor macro SPF_PPSTAT prior to including spf.h (which includes ppstat.h). Fields enumerated in the macro are added after the ppstat.l section. There is no automatic way to initialize the driver’s PPS; the driver must initialize it at run-time after allocating it via pps_add_entry(). See the spproto example driver for more details.

The ppstat.l section of the PPS includes these fields:

```c
Spf_ppstat  pp_next;  /* Next pointer in the list */
void       *pp_pd;    /* Path descriptor pointer */
Dev_list   pp_updrvr, /* Dev_list of driver above */
Dev_list   pp_dndrvr, /* Dev_list of driver below */
Dev_list   pp_deventry; /* driver's Dev_list */
```

The next PPS associated with a given logical unit static storage.
The pointer to the path descriptor with which the PPS corresponds.
The device list pointer for the “up” driver for this PPS.
The device list pointer for the “down” driver for this PPS.
The device list pointer for this driver.

The ppstat.l library includes functions to create, search for, and delete PPS. The following table lists and describes the functions that compose the ppstat.l library.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pps_add_entry()</td>
<td>Delete Host Entry by Name</td>
</tr>
<tr>
<td>pps_chg_updrvr()</td>
<td>Change the “up” Driver</td>
</tr>
<tr>
<td>pps_del_entry()</td>
<td>Delete a PPS</td>
</tr>
<tr>
<td>pps_find_entry()</td>
<td>Find a PPS</td>
</tr>
<tr>
<td>pps_find_entry_by_offset()</td>
<td>Find a PPS by a Key</td>
</tr>
</tbody>
</table>
**Syntax**

```c
#include <SPF/spf.h>
error_code
pps_add_entry(Dev_list deeventry, Spf_ppstat *ret_pps);
```

**Libraries**

`ppstat.l`

**Description**

`pps_add_entry()` allocates a PPS structure, clears the driver’s section to zero, and adds the PPS to the list of PPS structures for the specified logical unit static storage.

Prior to calling `pps_add_entry()`, the driver must initialize the `lu_pps_size` field of the logical unit static storage. In most cases, `sizeof(spf_ppstat)` should be used as the value.

The driver's section of the PPS will be cleared to zero by `pps_add_entry()`. Thus, it must be initialized at run-time after calling `pps_add_entry`.

**Parameters**

- `deeventry`  
  Pointer to the device list entry currently in use.

- `ret_pps`  
  Pointer to a pointer to a PPS structure. If this parameter is non-NULL, `pps_add_entry` will fill it in with a pointer to the new PPS.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** System

**Errors**

- `EOS_NGRAM`  
  Insufficient RAM.

- `EOS_PARAM`  
  Bad parameter (e.g. `lu_pps_size` is zero).
Syntax
#include <SPF/spf.h>

type error_code

pps_chg_updrvr(Spf_lustat lustat,
               Spf_pdstat pd,
               Dev_list new_updrvr);

Libraries
ppstat.1

Description

pps_chg_updrvr() changes the “up” driver (pp_updrvr field) for the PPS associated with the specified path descriptor. It searches the specified logical unit static storage for the PPS associated with the specified path descriptor and changes the “up” driver field of that PPS.

Parameters

lustat
    Current logical unit static storage.

pd
    The path descriptor associated with the PPS for which to change the “up” driver.

new_updrvr
    The new “up” driver device list pointer.

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              System

Errors

EOS_PPS_NOTFND       No PPS found for specified path descriptor.
pps_del_entry()

Delete a PPS

Syntax
#include <SPF/spf.h>
error_code
pps_find_entry(Spf_lustat lustat, Spf_pdstat pd);

Libraries
ppstat.1

Description
pps_del_entry() removes the PPS structure currently associated with the specified path descriptor from the PPS list for the specified logical unit static storage. The memory is returned to the operating system.

Parameters
lustat
   Pointer to the logical unit static storage from which to delete the PPS entry.

pd
   Pointer to the path descriptor with which the PPS to delete is associated.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              System

Errors
EOS_PPS_NOTFND      No PPS could be found for the specified path descriptor.
EOS_DAMAGE          _os_srtmem() error.
**Syntax**

```c
#include <SPF/spf.h>
error_code
pps_del_entry( Spf_lustat lustat,
              Spf_pdstat pd,
              Spf_ppstat *ret_pps);
```

**Libraries**

`ppstat.l`

**Description**

`pps_find_entry()` searches the PPS list on the specified logical unit static storage for a PPS structure associated with the specified path descriptor. A pointer to the matching PPS is returned at `ret_pps`. If none is found, the pointer at `ret_pps` will be set to NULL.

**Parameters**

- `lustat`  
  Pointer to the logical unit static storage to search.

- `pd`  
  Pointer to path descriptor with which the located PPS is associated.

- `ret_pps`  
  Pointer to a pointer to a PPS structure to fill in with the located PPS structure pointer.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** System

**Errors**

- **EOS_PPS_NOTFND**  
  No PPS found for specified path descriptor.
**Syntax**

```c
#include <SPF/spf.h>

error_code
pps_find_entry_by_offset(Spf_lustat lustat,
                         u_int16  offset,
                         u_int16  size,
                         void     *cmp_val,
                         Spf_ppstat *ret_pps);
```

**Libraries**

ppstat.l

**Description**

This function searches the PPS list on the specified logical unit static storage for a PPS structure containing a specified byte pattern at a specified offset. This function might be used to search via a key in the driver’s section of the PPS. A pointer to the matching PPS is returned at ret_pps. If none is found, the pointer at ret_pps will be set to NULL. For example, these two calls perform the same function:

```c
err = pps_find_entry(lustat, pd, &found);
err = pps_find_entry_by_offset(lustat,
                             offsetof(spf_ppstat, pp_pd), sizeof(pd), &pd,&found);
```

**Parameters**

- **lustat**
  - Pointer to the logical unit static storage to search.

- **offset**
  - The offset into the PPS to the key.

- **size**
  - The number of bytes in the key.

- **cmp_val**
  - A pointer to the key to search for.

- **ret_pps**
  - address of a pointer where the located PPS structure pointer is stored.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** System

**Errors**

- **EOS_PPS_NOTFND**
  - No PPS could be found for the specified path descriptor.
ITEM Library Functions

This section contains descriptions, in alphabetical order, of the telecommunications Application Programming Interface (API) functions in the Integrated Telephony Environment for Multimedia (ITEM) library.

The following table lists and describes the functions that compose the ITEM library.

Table 2-10. ITEM Library Functions and Descriptions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ite_ctl_addrset()</td>
<td>Sets ITEM Address Information</td>
</tr>
<tr>
<td>ite_ctl_answer()</td>
<td>Answers Incoming Call</td>
</tr>
<tr>
<td>ite_ctl_connect()</td>
<td>Establishes an End-to-end Connection</td>
</tr>
<tr>
<td>ite_ctl_connstat()</td>
<td>Returns Device Type Status Information</td>
</tr>
<tr>
<td>ite_ctl_disconnect()</td>
<td>Disconnects End-to-end Connection</td>
</tr>
<tr>
<td>ite_ctl_rcvrasgn()</td>
<td>Sets Up Path for Incoming Calls</td>
</tr>
<tr>
<td>ite_ctl_rcvrrmv()</td>
<td>Removes Notification Request</td>
</tr>
<tr>
<td>ite_data_avail_asgn()</td>
<td>Notifies Path of Incoming Data</td>
</tr>
<tr>
<td>ite_data_avail_rmv()</td>
<td>Removes Notification Request</td>
</tr>
<tr>
<td>ite_data_read()</td>
<td>Reads From a Path</td>
</tr>
<tr>
<td>ite_data_readmbuf()</td>
<td>Gets mbuf Data</td>
</tr>
<tr>
<td>ite_data_ready()</td>
<td>Returns Number of Bytes Available</td>
</tr>
<tr>
<td>ite_data_recvfrom()</td>
<td>Receive Data from a Specified Location</td>
</tr>
<tr>
<td>ite_data_sendto()</td>
<td>Send Data to a Specified Location</td>
</tr>
<tr>
<td>ite_data_write()</td>
<td>Writes Data to Packet</td>
</tr>
<tr>
<td>ite_data_writembuf()</td>
<td>Writes mbuf-packet</td>
</tr>
<tr>
<td>ite_dev_attach()</td>
<td>Attaches a Device</td>
</tr>
<tr>
<td>ite_dev_detach()</td>
<td>Detaches Device</td>
</tr>
<tr>
<td>ite_dev_getmode()</td>
<td>Gets Device Mode</td>
</tr>
<tr>
<td>ite_dev_getname()</td>
<td>Gets Device Name</td>
</tr>
<tr>
<td>ite_dev_gettype()</td>
<td>Gets Device Type</td>
</tr>
<tr>
<td>ite_dev_setmode()</td>
<td>Sets Device Mode</td>
</tr>
<tr>
<td>ite_fehangup_asgn()</td>
<td>Registers Caller for Notification of Far End Disconnect</td>
</tr>
<tr>
<td>ite_fehangup_rmv()</td>
<td>Removes Request for Far-end Hang-up</td>
</tr>
<tr>
<td>ite_linkdown_asgn()</td>
<td>Notifies Caller of Link Failure</td>
</tr>
<tr>
<td>ite_linkdown_rmv()</td>
<td>Removes Linkdown Notification Assignment</td>
</tr>
<tr>
<td>ite_linkup_asgn()</td>
<td>Notifies Caller of Link Failure</td>
</tr>
<tr>
<td>ite_linkup_rmv()</td>
<td>Removes Linkup Notification Assignment</td>
</tr>
<tr>
<td>ite_path_clone()</td>
<td>Duplicates a Path - Independent Connection</td>
</tr>
<tr>
<td>ite_path_close()</td>
<td>Closes a Path</td>
</tr>
<tr>
<td>ite_path_dup()</td>
<td>Duplicates Path and Shares Connection</td>
</tr>
<tr>
<td>ite_path_open()</td>
<td>Opens a Path</td>
</tr>
</tbody>
</table>
## Table 2-10. ITEM Library Functions and Descriptions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ite_path_pop()</td>
<td>Removes Driver from Top of Stack</td>
</tr>
<tr>
<td>ite_path_profileget()</td>
<td>Get Path Profile</td>
</tr>
<tr>
<td>ite_path_profileset()</td>
<td>Set Path Profile</td>
</tr>
<tr>
<td>ite_path_push()</td>
<td>Pushes Protocol or Hardware-Driver Onto Path</td>
</tr>
</tbody>
</table>
ite_ctl_addrset()

Sets ITEM Address Information

Syntax

```c
#include <SPF/item.h>
error_code ite_ctl_addrset(
    ath_id        path,
    addr_type     *our_num,
    addr_type     *their_num);
```

Libraries

item.l

Description

ite_ctl_addrset() allows you to set the address information for the ITEM path. Specifically, it sets the dev_ournum and dev_theirnum structures in the item path descriptor. If our_num or their_num is set to NULL, the corresponding value of dev_ournum and dev_theirnum respectively is left unchanged.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

path contains a handle identifying the input/output (I/O) path. This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

For in-band protocols, the path is also used to read and write data. For out-of-band protocols, the path is used only as the signalling path; it is not used to read and write data.

our_num points to the caller-allocated addr_type structure set up to represent the new address information for your device.

their_num points to the caller-allocated addr_type structure. It is set up to represent the new address of the far-end device with which to communicate.

Indirect Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_ILLPRM</td>
<td>addr_size field in addr_type structure is greater than 32 bytes.</td>
</tr>
<tr>
<td>EOS_BPNUM</td>
<td>Returned when the path number is invalid.</td>
</tr>
</tbody>
</table>
ITE_CTL_ADDRSET() returns the following errors:

- **EOS_PTHLOST**  Returned when the path was lost and is no longer valid.
- **EOS_PPS_NTFND** Returned when the driver cannot find its local path storage for the path_id passed in.

Other errors may be returned by SoftStax® drivers.

Refer to the description of item.h in the Using SoftStax manual for information about the addr_type structure.
ite_ctl_answer()

Answers Incoming Call

Syntax

```c
#include <SPF/item.h>
error_code ite_ctl_answer(
    path_id         path,
    ite_cctl_pb     *ccpb,
    notify_type     *npb);
```

Libraries

item.l

Description

ite_ctl_answer() allows a process to answer an incoming call on the specified path. The path must have previously performed an ite_ctl_rcvrasgn() call to enable notification of incoming calls.

Alternatively, the application can continuously poll using the ite_ctl_connstat() call until the dev_callstate field equals ITE_CS_INCALL.

If the npb parameter is NULL, control is returned to the caller after the call control procedure puts this connection into the active state, indicating an end-to-end connection is being established.

If a notification parameter block is passed in, the driver sends notification when the end-to-end connection has been confirmed by the network. In either case, before answering, the ite_ctl_connstat() call can be used to screen the call or get the display information, if there is any, for the incoming call (that is, caller ID).

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `path`: contains a handle identifying the I/O path. This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

  For in-band protocols, the path is also used to read and write data. For out-of-band protocols, the path is used only as the signalling path; it is not used to read and write data.
**ccpb** points to the call control parameter block. This parameter block provides additional information from the application to the driver responsible for performing the answer. If no additional information is needed, the application may pass **NULL** for this pointer.

**npb** points to the notification parameter block, telling ITEM the type of notification the caller requests when the connection is established.

Refer to information on `notify_type` in the *Using SoftStax* manual for information about setting up the notification parameter block.

### Indirect Errors

- **EOS_BPNUM**
  - Returned when the path number is invalid.

- **EOS_PPS_NOTFND**
  - Returned when the driver cannot find its local path storage for the `path_id` passed in.

- **EOS_PTHLOST**
  - Returned when the path was lost and is no longer valid.

- **EOS_TSTATE**
  - Path is in the wrong state to answer incoming call.

- **EOS_UNKSVC**
  - Returned when drivers for connection-oriented ITEM calls (connect, disconnect, or answer) are not connected.

### See Also

- `ite_ctl_addrset()`
- `ite_ctl_connect()`
- `ite_ctl_connstat()`
- `ite_ctl_disconnect()`
ite_ctl_connect()
Establishes an End-to-end Connection

Syntax
#include <SPF/item.h>
error_code ite_ctl_connect(
    path_id        path,
    addr_type      *ournum,
    addr_type      *theirnum,
    notify_type    *npb);

Libraries
item.l

Description
ite_ctl_connect() sets the dev_ournum and dev_theirnum structures in the item path descriptor, to the addr_type structures that you passed in. This is only done if the ournum or theirnum pointer is not NULL. Then, this function establishes an end-to-end connection from the ITEM device referenced by path to the far-end device with the address found in the dev_theirnum structure of your device. If the ournum and/or theirnum pointers passed in are NULL, ITEM uses the default addresses found in the path descriptor.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
path contains a handle identifying the I/O path. This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

For in-band protocols, the path is also used to read and write data. For out-of-band protocols, the path is used only as the signalling path; it is not used to read and write data.

ournum points to the caller-allocated addr_type structure set up to represent the new local address information for your device.
theirnum points to the caller-allocated addr_type structure. It is set up to represent the new remote address of the far-end device with which to communicate.

npb points to the notification parameter block. This tells ITEM the type of notification the caller requests upon establishing a connection.

Refer to information on notify_type in the Using SoftStax manual for information about setting up the notification parameter block.

Direct Errors
EOS_ILLPRM The notify parameter block (npb) passed in is NULL.

Indirect Errors
EOS_DEVBSY Returned when trying to connect to a path that already has, or is establishing, a connection. (The device_type cellstate is ITE_CS_CONNECT, ITE_CS_ACTIVE, or ITE_CS_CONNTERM). In order for a connection to be successful, the path must be in the ITE_CS_IDLE state.
EOPNOTSUPP Returned when trying to connect on a connectionless protocol.
EOS_UNKSVC Returned if no protocol on the stack performs cell control functions.
EOS_BPNUM Returned when the path number is invalid.
EOS_PTHLOST Returned when the path was lost and is no longer valid.

See Also
ite_ctl_addrset()
icte_ctl_answer()
icte_ctl_connstat()
icte_ctl_disconnect()
ite_ctl_connstat()

Returns Device Type Status Information

Syntax

```c
#include <SPF/item.h>
error_code ite_ctl_connstat(
    path_id         path,
    device_type     *dev_info);
```

Libraries

item.l

Description

ite_ctl_connstat() returns a copy of the dev_type structure for the specified path. This structure contains information about the our-end and far-end connection addresses, call state, display information, and the network type of the device.

Refer to device_type in Using SoftStax for information about its fields.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- path contains a handle identifying the I/O path. This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket(). For in-band protocols, the path is also used to read and write data. For out-of-band protocols, the path is used only as the signalling path; it is not used to read and write data.

Indirect Errors

- EOS_BPNUM Returned when the path number is invalid.
- EOS_BPADD Returned if dev_info is NULL or points to memory not owned by the process. This error is only returned on systems with SSM (system security module) running.
- EOS_PTHLOST Returned when the path was lost and is no longer valid.

Refer to item.h in Using SoftStax for information about the dev_type.

See Also

- ite_ctl_addrset()
ite_ctl_disconnect()
Disconnects End-to-end Connection

Syntax
#include <SPF/item.h>
error_code ite_ctl_disconnect(
    path_id         path,
    ite_cctl_pb     *ccpb);

Libraries
item.l

Description
ite_ctl_disconnect() disconnects the end-to-end connection established by
ite_ctl_connect().

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
path contains a handle identifying the I/O path.
This handle is usually obtained from calls such as
ite_path_open(), _os_open(), and socket(). For in-band
protocols, the path is also used to read and write data. For
out-of-band protocols, the path is used only as the signalling
path; it is not used to read and write data.

ccpb points to the call control parameter block.
This parameter block provides additional information from
the application to the driver for performing the
disconnection. If no additional information is needed, the
application can pass NULL for this pointer.

Indirect Errors
EOS_UNKSVC Returned if no protocol on the stack performs cell control
functions.
EOS_BPNUM Returned when the path number is invalid.
EOS_PTHLOST Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND Returned when the driver cannot find its local path storage
for the path_id passed in.

See Also
ite_ctl_connect()
ite_ctl_rcvrasgn()  
Sets Up Path for Incoming Calls

Syntax

```c
#include <SPF/item.h>
error_code ite_ctl_rcvrasgn(
    path_id         path,
    addr_type       *their_num,
    notify_type     *npb);
```

Libraries

item.l

Description

ite_ctl_rcvrasgn() sets up the calling process to receive notification of an incoming call. If a notification has already been registered on this path, this call returns EOS_DEVBSY. You can set up only one process per path to receive incoming calls.

If `their_num` is NULL, the path receives a notification for any incoming call.

If `their_num` is not NULL, the path only receives notification for an incoming call with a calling address matching the `their_num` address.

Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

Parameters

- **path** contains a handle identifying the I/O path. This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.

  For in-band protocols, the path is also used to read and write data. For out-of-band protocols, the path is used only as the signalling path; it is not used to read and write data.

- **their_num** contains either a NULL value, or points to the `addr_type` structure.

  The caller allocates and sets up this structure to receive incoming calls from a specified address. If NULL is used, the application receives notification on any incoming call.

- **npb** points to the notification parameter block structure ITEM uses to send notification to the caller.
Direct Errors

EOS_ILLPRM  Returned when npb is NULL.

Indirect Errors

EOS_ILLPRM  Returned if the addr_size parameter in the their_num structure is greater than 32.
EOS_DEVBSY  Returned if any process has already registered for receiving notification of an incoming call.
EOS_BPNUM  Returned when the path number is invalid.
EOS_PTHLOST  Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND  Returned when the driver cannot find its local path storage for the path_id passed in.

See Also

ite_ctl_rcvrrmv()
ite_ctl_rcvrrmv()

Removes Notification Request

Syntax
#include <SPF/item.h>

error_code ite_ctl_rcvrrmv(path_id path);

Libraries
item.l

Description
ite_ctl_rcvrrmv() removes the notification request created with
ite_ctl_rcvrasgn(). This call resolves successfully even if the application executes
the call without a prior notification assignment.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
path contains a handle identifying the I/O path. This handle is
usually obtained from calls such as ite_path_open(),
_os_open(), and socket().

For in-band protocols, the path is also used to read and write
data. For out-of-band protocols, the path is used only as the
signalling path; it is not used to read and write data.

Indirect Errors
EOS_BPNUM Returned on a bad path number.
EOS_BPNUM Returned when the path number is invalid.
EOS_PTHLOST Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND Returned when the driver cannot find its local path storage
for the path_id passed in.

See Also
ite_ctl_rcvrasgn()
**ite_data_avail_asgn()**

**Notifies Path of Incoming Data**

**Syntax**

```c
#include <SPF/item.h>
error_code ite_data_avail_asgn(
    path_id         path,
    notify_type     *npb);
```

**Libraries**

item.l

**Description**

`ite_data_avail_asgn()` sets up the calling process to receive a notification when incoming data is available for the specified path.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **path** contains a handle identifying the I/O path. This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.

  For in-band protocols, the path is also used to read and write data. For out-of-band protocols, the path is used only as the signalling path; it is not used to read and write data.

- **npb** points to the notification parameter block structure ITEM uses to send notification to the caller.

**Direct Errors**

- **EOS_ILLPRM** Returned when `npb` is NULL or, if requesting an event, the `ntfy_evid` is 0.

**Indirect Errors**

- **EOS_BPNUM** Returned when the path number is invalid.
- **EOS_PTHLOST** Returned when the path is lost and no longer valid.
- **EOS_PPS_NOTFND** Returned when the driver cannot find its local path storage for the `path_id` passed in.
See Also

ite_data_avail_rmv()
ite_data_ready()
_os_ss_sendsig() in the OS-9 Technical Manual
ite\_data\_avail\_rmv()

Removes Notification Request

Syntax

```c
#include <SPF/item.h>
error\_code ite\_data\_avail\_rmv(path\_id path);
```

Libraries

item.l

Description

ite\_data\_avail\_rmv() removes the notification request created with ite\_data\_avail\_asgn(). This call resolves successfully even if the application executes it without a prior notification assignment.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

path contains a handle identifying the I/O path. This handle is usually obtained from calls such as ite\_path\_open(), _os\_open(), and socket().

Indirect Errors

- **EOS\_BPNUM**: Returned when the path number is invalid.
- **EOS\_PTHLOST**: Returned when the path was lost and is no longer valid.
- **EOS\_PPS\_NOTFND**: Returned when the driver cannot find its local path storage for the path\_id passed in.

See Also

- ite\_data\_avail\_asgn()
- _os\_ss\_relea() in the OS-9 Technical Manual
ite_data_read()

Reads From a Path

Syntax

```
#include <SPF/item.h>

error_code ite_data_read(
    path_id    path,
    void       *buffer,
    u_int32    *count);
```

Libraries

item.l

Description

ite_data_read() performs a read on a path. Data is removed from the path’s receive queue and passed back to the caller in buffer. The read operation depends on the I/O options of the path, (pd_ispacket, pd_ioasync, and pd_iotime) which may be changed via the _os_ss_popt() call.

Refer to the Using SoftStax manual for more information about pd_ispacket, pd_ioasync, and pd_iotime.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- path contains a handle identifying the I/O path. This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().
- buffer points to the caller-allocated buffer in which to place the data.
  SoftStax copies the data from the receive queue into this buffer.
- count points to the number of bytes to read.
  When the call is completed, the address that is pointed to is updated with the number of bytes SoftStax copies into the buffer.
Indirect Errors

EOS_DEVBSY Returned when another process is already waiting for incoming data on this path.

EOS_NOTRDY Returned when the SoftStax driver’s lu_ioenabled flag is 0. This error indicates either that the protocol stack contains a driver that is not initialized or the end-to-end protocol has an error causing a break in the end-to-end protocol link.

EOS_SIGNAL Returned when a fatal signal is received.

ETIMEOUT Returned when the read request timed out before completion.

EWOULDBLOCK Returned when a read request is made, but there is currently not enough data available to actually read. This error occurs only if the IO_READ_ASYNC bit is set in the pd_ioasync byte in the path options, indicating the read side of the path is in asynchronous mode.

E_BPADDR Returned if buffer is NULL or does not point to memory owned by the calling process.

EOS_RXMB_ERR Returned when data being read into the application buffer is flagged as incorrect by the received data hardware. This could happen as a result of bad CRC, overflow, abort sequence, or bad hardware. At this point, the application should request hardware statistics to discover the source of the error. The data is returned through the read call. The hardware error occurred somewhere in the current received buffer.

EOS_BFNUM Returned when the path number is invalid.

EOS_PTHLOST Returned when the path is lost and no longer valid.

EOS_PPS_NOTFND Returned when the driver cannot find its local path storage for the path_id passed in.

See Also

_os_gs_popt() and _os_ss_popt() in the OS-9 Technical Manual
ite_data_readmbuf()

Gets mbuf Data

Syntax

```c
#include <SPF/item.h>

error_code ite_data_readmbuf(
    path_id     path,
    mbuf        *mb_ptr);
```

Libraries

item.l

Description

ite_data_readmbuf() allows the caller to get the buffer SoftStax uses to store the mbuf (incoming packets) instead of passing in a user buffer with data copied into it. This call returns one mbuf packet chain in mb_ptr. This call facilitates true zero copy reads for faster throughput. SoftStax gives the caller permissions to access the returned mbuf packet chain.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

`path` contains a handle identifying the I/O path.
This handle is usually obtained from calls such as ite_path_open(), os_open(), and socket().

`mb_ptr` contains the pointer to the mbuf packet chain.

Indirect Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_DEVBSY</td>
<td>Returned when another process is already waiting for incoming data on this path.</td>
</tr>
<tr>
<td>EOS_NOTRDY</td>
<td>Returned when the SoftStax driver’s lu_ioenabled flag is 0. This error indicates that the protocol stack contains a driver that is not initialized or the end-to-end protocol has an error causing a break in the end-to-end protocol link.</td>
</tr>
<tr>
<td>EOS_SIGNAL</td>
<td>Returned when a fatal signal is received.</td>
</tr>
<tr>
<td>ETIMEOUT</td>
<td>Returned when the read request timed out before completion.</td>
</tr>
</tbody>
</table>
EWOULDBLOCK  Returned when a read request is made but there is currently not enough data available to read. This error only occurs if the IO_READ_ASYNC bit is set in the pd_ioasync byte in the path options, indicating the read side of the path is in asynchronous mode.

EOS_BPNUM  Returned when the path number is invalid.

EOS_PTHLOSS  Returned when the path is lost and no longer valid.

EOS_PPS_NOTFND  Returned when the driver cannot find its local path storage for the path_id passed in.

See Also

ite_data_read()
ite_data_writembuf()
the mbuf facility in the Using SoftStax manual
ite_data_ready()

Returns Number of Bytes Available

Syntax

```c
#include <SPF/item.h>

error_code ite_data_ready(
    path_id      path,
    u_int32      *avail_count);
```

Libraries

`item.l`

Description

ite_data_ready() counts the number of bytes on the specified path that are available for reading. An EOS_NOTRDY error is returned if there is no data available.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `path` contains a handle identifying the I/O path. This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.
- `avail_count` Points to a `u_int32` field where the number of bytes available for reading is returned.

Indirect Errors

- `EOS_NOTRDY` Returned if no data is ready to read.
- `EOS_BPNUM` Returned when the path number is invalid.
- `EOS_PTHLOST` Returned when the path is lost and no longer valid.
- `EOS_PPS_NOTFND` Returned when the driver cannot find its local path storage for the `path_id` passed in.

See Also

`_os_gs_ready()` in the OS-9 Technical Manual
ite_data_recvfrom()
Receive Data from a Specified Location

Syntax

```c
#include <SPF/item.h>
error_code ite_data_recvfrom(
    path_id     path,
    void        *buffer,
    u_int32     size,
    u_int32     flags,
    addr_type   *recvfrom_addr);
```

Libraries

item.l

Description

ite_data_recvfrom() allows the caller to send data to a particular remote location. The call might block if the IO_WRITEASYNC bit in the pd_ioasync byte in the path options structure is set and there are no mbufs available in the mbuf pool.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `path` contains a handle identifying the I/O path. This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.
- `buffer` points to the data to be transmitted.
- `size` is the number of valid bytes of data in the buffer.
- `flags` is not used.
- `sendto_addr` points to the caller-allocated addr_type structure. It is set up to represent the remote address of the far-end device to which the caller is sending data.
Indirect Errors

ENOBDFS  Returned when both the IO_WRITE_ASYNC bit in the pd_ioasync byte is set and the transmit mbuf cannot be allocated for the data being written. This indicates that either the mbuf pool is empty or the system cannot allocate an mbuf large enough for this transport packet.

EOS_NOTRDY  Returned when the SoftStax driver’s lu_ioenabled flag is 0, indicating the protocol stack contains a driver that is not initialized or the end-to-end protocol has an error causing a break in the end-to-end protocol link.

EOS_PPS_NOTFND  Returned when the driver cannot find its local path storage for the path_id passed in.

Other errors may be returned by the SoftStax drivers.
**Syntax**

```c
#include <SPF/item.h>
error_code ite_data_sendto(
    path_id      path,
    void         *buffer,
    u_int32      size,
    u_int32      flags,
    addr_type    *sendto_addr);
```

**Libraries**

`item.l`

**Description**

`ite_data_sendto()` allows the caller to send data to a particular remote location. The call may block if the `IO_WRITE_ASYNCH` bit in the `pd_ioasync` byte in the path options structure is set and there are no mbufs available in the mbuf pool.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **path** contains a handle identifying the I/O path. This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.
- **buffer** points to the data to be transmitted.
- **size** is the number of valid bytes of data in the buffer.
- **flags** is not used.
- **sendto_addr** points to the caller-allocated `addr_type` structure. It is set up to represent the remote address of the far-end device to which the caller is sending data.
Indirect Errors

ENOBIFS Returned when both the IO_WRITE_ASYNC bit in the pd_ioasync byte is set and the transmit mbuf cannot be allocated for the data being written. This indicated that either the mbuf pool is empty or the system cannot allocate an mbuf large enough for this transport packet.

EOS_NOTR Dy Returned when the SoftStax driver’s lu_ioreabled flag is 0, indicating the protocol stack contains a driver that is not initialized or the end-to-end protocol has an error causing a break in the end-to-end protocol link.

EOS_PPS_NOTFND Returned when the driver cannot find its local path storage for the path_id passed in.

Other errors may be returned by the SoftStax drivers.
ite_data_write()

Writes Data to Packet

Syntax

```c
#include <SPF/item.h>
error_code ite_data_write(
    path_id     path,
    void        *buffer,
    u_int32     *count);
```

Libraries

item.l

Description

ite_data_write() causes SoftStax to create a packet out of the buffer that is to be written. In addition, it sends the packet down the protocol chain for the path until the hardware driver queues it up for transmission.

The call might block if the IO_WRITEASYNC bit in the pd_ioasync byte in the path options structure is set and there are no mbufs available in the mbuf pool.

A return from this call does not ensure the data has been transmitted. It only indicates that the data has been queued for transmission by the device.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- path contains a handle identifying the I/O path.
  This handle is usually obtained from calls such as ite_path_open(), os_open(), and socket().
- buffer points to the data to be transmitted.
- count points to the number of valid bytes of data in the buffer.

Indirect Errors

- ENOBUFS Returned when both the IO_WRITEASYNC bit in the pd_ioasync byte is set and the transmit mbuf cannot be allocated for the data being written. This indicated that either the mbuf pool is empty or the system cannot allocate an mbuf large enough for this transport packet.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_NOTRDY</td>
<td>Returned when the SoftStax driver’s <code>lu_ioenabled</code> flag is 0, indicating the protocol stack contains a driver that is not initialized or the end-to-end protocol has an error causing a break in the end-to-end protocol link.</td>
</tr>
<tr>
<td>E_BPADDR</td>
<td>Returned if buffer is NULL.</td>
</tr>
<tr>
<td>EOS_BPNUM</td>
<td>Returned when the path number is invalid.</td>
</tr>
<tr>
<td>EOS_PTHLOST</td>
<td>Returned when the path is lost and no longer valid.</td>
</tr>
<tr>
<td>EOS_PPS_NOTFND</td>
<td>Returned when the driver cannot find its local path storage for the <code>path_id</code> passed in.</td>
</tr>
</tbody>
</table>
ite_data_writembuf()

Writes mbuf-packet

Syntax

```c
#include <SPF/item.h>
error_code ite_data_writembuf(
    path_id    path,
    mbuf       mb_ptr);
```

Libraries

item.1

Description

ite_data_writembuf() accepts an mbuf for transmission through the protocol stack and out to the hardware device. For best efficiency, the caller should save enough space at the beginning of the mbuf to allow the protocol stack to add headers to the same mbuf passed in. The caller can obtain the number of bytes the stack needs by getting the path options and looking at the `pd_txoffset` field. Otherwise, the protocol must find another mbuf in which to place the header information and chain those packets together.

Drivers that cannot operate correctly on mbuf chains are unable to process this call if there is not enough space for all headers at the beginning of the mbuf.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `path` contains a handle identifying the I/O path. This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.

- `mb_ptr` points to the mbuf packet chain that is to be transmitted.

Indirect Errors

- `EOS_BPNUM` Returned when the path number is invalid.
- `EOS_PTHLOST` Returned when the path is lost and no longer valid.
- `EOS_PPS_NOTFND` Returned when the driver cannot find its local path storage for the `path_id` passed in.

See Also

- `ite_data_readmbuf()` `ite_data_write()`
- the mbuf facility in the *Using SoftStax* manual
ite_dev_attach()
Attaches a Device

Syntax

```c
#include <SPF/item.h>
error_code ite_dev_attach(
    char        *name,
    u_int32     mode,
    u_int32     *handle);
```

Libraries

item.l

Description

ite_dev_attach() performs a simple attach of a device into ITEM.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- **name**: points to the name of the device descriptor with which to attach.
- **mode**: contains the mode in which to open the device. Recommended default value for mode is `FAM_READ` or `FAM_WRITE`.
- **handle**: points to the unique ID returned by ITEM. It identifies this device.

Indirect Errors

- **EOS_UNKSVC**: Returned when the `sysmbuf` utility is not installed.
- **EOS_MNF**: Returned when the driver, descriptor, or file manager is not loaded into memory.

Drivers may return their own error codes. Refer to the specific driver’s description for more information.

See Also

- `ite_dev_detach()`
- `_os_attach()` in the *Ultra C Library Reference* manual

OS-9 Network Programming Reference 224
ite_dev_detach()
Detaches Device

Syntax
#include <SPF/item.h>
error_code ite_dev_detach(u_int32 handle);

Libraries
item.l

Description
ite_dev_detach() allows you to detach a device from ITEM. If the attach count is greater than 1 (was attached multiple times by multiple applications), only the attach count of the device is decremented; thus, the device remains in the set of active devices used by ITEM. When the attach count is 1 (the last application detaches from the ITEM device), the device is removed from the set of initialized devices used by ITEM.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
handle
contains the unique ID returned by ITEM. It identifies this device.

Drivers may return their own error codes. Refer to the specific driver’s description for more information.

See Also
ite_dev_attach()
_os_detach() in the Ultra C Library Reference manual
ite_dev_getmode()

Gets Device Mode

Syntax

```
#include <SPF/item.h>

error_code ite_dev_getmode(
    path_id      path,
    u_int16      *mode);
```

Libraries

item.l

Description

ite_dev_getmode() allows you to determine the mode of the device using the specified path.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `path` contains a handle identifying the I/O path.
  This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

- `mode` points to where SoftStax returns the mode of the device the path is using.
  The possible bit field values for mode are: FAM_READ, FAM_WRITE, or FAM_NONSHARE.

Indirect Errors

- EOS_BPNUM Returned when the path number is invalid.
- EOS_PTHLOST Returned when the path is lost and no longer valid.

See Also

- modes.h contains macros for the mode bit fields.
ite_dev_getname()  
Gets Device Name

Syntax

#include <SPF/item.h>

type error_code ite_dev_getname(
    path_id      path,
    char         *name);

Libraries

item.l

Description

ite_dev_getname() allows you to determine the name of the device using the specified path.

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User
Threads:            Safe

Parameters

path
    contains a handle identifying the I/O path.
    This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

name
    points to the user allocated buffer where SoftStax puts the device name string.
    In OS-9, the buffer pointed to by name must be at least 64 bytes to accommodate the device name. In OS-9 for 68K processors, the buffer should be at least 32 bytes.

Indirect Errors

EOS_BPNUM       Returned when the path number is invalid.
EOS_PTHLOST     Returned when the path is lost and no longer valid.
ite_dev_gettype()

Gets Device Type

Syntax
#include <SPF/item.h>
error_code ite_dev_gettype(
    path_id       path,
    u_char       *type_in,
    u_char       *type_out);

Libraries
item.l

Description
ite_dev_gettype() allows you to determine the type of the device using the specified path.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

path
contains a handle identifying the I/O path.
This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

type_in
points to where SoftStax puts the type of the receive-side of the ITEM device.

type_out
points to where SoftStax places the type of the transmit-side of the ITEM device.

Table 2-11 lists the different device types.

Table 2-11. Device Types

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITE_NET_NONE</td>
<td>If only one side has this value, the device is unidirectional.</td>
</tr>
<tr>
<td>ITE_NET_CTL</td>
<td>Set-top box control channel device.</td>
</tr>
<tr>
<td>ITE_NET_DATA</td>
<td>Set-top box data channel device.</td>
</tr>
</tbody>
</table>
Table 2-11. Device Types (Continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITE_NET_MPEG2</td>
<td>MPEG-2 device.</td>
</tr>
<tr>
<td>ITE_NET_CHMGR</td>
<td>Set-top box channel management device.</td>
</tr>
<tr>
<td>ITE_NET_OOB</td>
<td>Out-of-band code signalling device.</td>
</tr>
<tr>
<td>ITE_NET_ANY</td>
<td>Generic network device. No specific payload is required.</td>
</tr>
<tr>
<td>ITE_NET_VIPDIR</td>
<td>Video Information Provider Directory</td>
</tr>
<tr>
<td>ITE_NET_SESCTL</td>
<td>Session control device.</td>
</tr>
</tbody>
</table>

Indirect Errors

EOS_BPNUM Returned when the path number is invalid.

EOS_PTHLOST Returned when the path is lost and is no longer valid.

Refer to the information on item.h and the Using SoftStax manual for the device_type structure.
ite_dev_setmode()

Sets Device Mode

Syntax

```c
#include <SPF/item.h>
error_code ite_dev_setmode(
    path_id      path,
    u_int16      mode);
```

Libraries

item.l

Description

ite_dev_setmode() allows you to set the device mode.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

path

contains a handle identifying the I/O path.
This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

mode

contains the mode to set. Bit field values for mode include the following:

- FAM_READ
- FAM_WRITE
- FAM_NONSHARE

Indirect Errors

- EOS_BPNUM: Returned when the path number is invalid.
- EOS_PTHLOST: Returned when the path is lost and no longer valid.
- EOS_PPS_NOTFND: Returned when the driver cannot find its local path storage for the path_id passed in.

See Also

- ite_dev_getmode()
- item.h and Using SoftStax manual for the device_type structure
- modes.h for the macros that define the mode bits
ite_fehangup_asgn()

Registers Caller for Notification of Far End Disconnect

Syntax

```
#include <SPF/item.h>

error_code ite_fehangup_asgn(
    path_id       path,
    notify_type   *npb);
```

Libraries

item.l

Description

ite_fehangup_asgn() registers the caller for notification of disconnection by the far-end.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- **path**
  - contains a handle identifying the I/O path.
  - This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

- **npb**
  - Points to the notification parameter block structure ITEM uses to send notification to the caller.

Indirect Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_ILLFRM</td>
<td>Returned when npb is NULL.</td>
</tr>
<tr>
<td>EOS_DEVBSY</td>
<td>Returned if another process has already registered for this notification on the path.</td>
</tr>
<tr>
<td>EOS_BPNUM</td>
<td>Returned when the path number is invalid.</td>
</tr>
<tr>
<td>EOS_PTHLOST</td>
<td>Returned when the path is lost and no longer valid.</td>
</tr>
<tr>
<td>EOS_PPS_NOTFND</td>
<td>Returned when the driver cannot find its local path storage for the path_id passed in.</td>
</tr>
</tbody>
</table>

See Also

ite_fehangup_rmv()
ite_fehangup_rmv()

Removes Request for Far-end Hang-up

Syntax
#include <SPF/item.h>
error_code ite_fehangup_rmv(path_id path);

Libraries
item.l

Description
ite_fehangup_rmv() removes the notification request for far-end disconnection. This call resolves successfully even if the application executes it without a prior notification assignment.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
path
    contains a handle identifying the I/O path.
    This handle is usually obtained from calls such as ite_path_open(),
    _os_open(), and socket().

Indirect Errors
EOS_BPNUM       Returned when the path number is invalid.
EOS_PTHLOST     Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND  Returned when the driver cannot find its local path storage
                for the path_id passed in.

See Also
ite_fehangup_asgn()
ite_linkdown_asgn()
Notifies Caller of Link Failure

Syntax
#include <SPF/item.h>
error_code ite_linkdown_asgn(
    path_id          path,
    notify_type      *npb);

Libraries
item.l

Description
ite_linkdown_asgn() tells SoftStax drivers to notify the caller if the link fails at any layer of the protocol stack.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
path
    contains a handle identifying the I/O path.
    This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket()..

npb
    points to the notification parameter block structure ITEM uses to send notification to the caller.

Indirect Errors
EOS_ILLPRM     Returned when npb is NULL.
EOS_DEVBSY     Returned if another process has already registered for this notification on the path.
EOS_BPNUM      Returned when the path number is invalid.
EOS_PTHLOST    Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND Returned when the driver cannot find its local path storage for the path_id passed in.

See Also
ite_linkdown_rmv()
**ite_linkdown_rmv()**

Removes Linkdown Notification Assignment

**Syntax**

```c
#include <SPF/item.h>
error_code ite_linkdown_rmv(path_id path);
```

**Libraries**

item.l

**Description**

`ite_linkdown_rmv()` removes the previous link failure notification assignment created with the `ite_linkdown_asgn()` call. This call returns successfully even if the application executes it without a prior notification assignment.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **path** contains a handle identifying the I/O path.
  - This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.

**Indirect Errors**

- **EOS_BPNUM** Returned when the path number is invalid.
- **EOS_PTHLOST** Returned when the path is lost and no longer valid.
- **EOS_PPS_NOTFND** Returned when the driver cannot find its local path storage for the `path_id` passed in.

**See Also**

`ite_linkdown_asgn()`
ite_linkup_asgn()
Notifies Caller of Link Failure

Syntax
#include <SPF/item.h>
error_code ite_linkup_asgn(
    path_id       path,
    notify_type   *npb);

Libraries
item.l

Description
ite_linkup_asgn() tells SoftStax drivers to notify the caller if the link fails at any layer of the protocol stack.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User
Threads:            Safe

Parameters
path
    contains a handle identifying the I/O path.
    This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

npb
    points to the notification parameter block structure ITEM uses to send notification to the caller.

Indirect Errors
EOS_ILLPRM          Returned when npb is NULL.
EOS_DEVBSY          Returned if another process has already registered for this notification on the path.
EOS_BPNUM           Returned when the path number is invalid.
EOS_PTHLOST         Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND      Returned when the driver cannot find its local path storage for the path_id passed in.

See Also
ite_linkup_rmv()
ite_linkup_rmv()

Removes Linkup Notification Assignment

Syntax
#include <SPF/item.h>
error_code ite_linkup_rmv(path_id path);

Libraries
item.l

Description
ite_linkup_rmv() removes the previous link failure notification assignment created with ite_linkup_asgn(). This call returns successfully even if the application executes it without prior notification.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
path contains a handle identifying I/O path. This handle is obtained from ite_path_open(),_os_open(), and socket().

Indirect Errors
EOS_BPNUM Returned when the path number is invalid.
EOS_PTHLOST Returned when path is no longer valid.
EOS_PPS_NOTFND Returned when the driver cannot find its local path storage for the path_id passed in.

See Also
ite_linkup_asgn()
**Syntax**

```c
#include <SPF/item.h>
error_code ite_path_clone(
    path_id    dup_path,
    path_id    *new_path,
    notify_type  *npb);
```

**Libraries**

item.l

**Description**

`ite_path_clone()` creates a new path with the same device type information, properties, and connection state as the original path. It assigns a new path identifier to the newly cloned path.

`ite_path_clone()` is similar to `ite_path_dup()`, except that if the original path has an active connection, the `ite_clone_path()` call creates a new connection to the same two endpoints as the original path. If the paths are duplicated (with `ite_path_dup()`), both paths share a connection. Also, unlike duplicated paths, cloned paths point to different SoftStax path descriptors.

If you are duplicating a connection-oriented device, this call uses the asynchronous `ite_ctl_connect()` call. In this case, the `ite_path_clone()` call becomes asynchronous and the caller is notified using the `npb` notification. Otherwise, `ite_path_clone()` is synchronous for connectionless devices.

Refer to `ite_ctl_connect()` for more information about `npb`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
ite_path_clone()

Parameters

dup_path
specifies the path to clone.

new_path
points to the location where the ID of the new path is returned.

A new connection is established between the ITEM device and the far-end address, as specified in the device_type structure for the original path. This ensures that the dup_path and new_path do not share a connection.

npb
points to the notification parameter block structure ITEM uses to send notification to the caller.

Refer to the Using SoftStax manual for information about the notify_type structure.

Indirect Errors

EOS_BPNUM
Returned when the path number is invalid.

EOS_PTHLOST
Returned when the path is lost and no longer valid.

EOS_DEVBSY
Returned when you are trying to clone a non-sharable device on the second open to the device. You can avoid this error by calling ite_dev_getmode() to make sure the path you are cloning is a sharable device.

See Also

ite_ctl_connect()
ite_path_dup()
ite_path_close()

Closes a Path

Syntax

```
#include <SPF/item.h>
error_code ite_path_close(path_id path);
```

Libraries

item.l

Description

ite_path_close() closes the specified path. If there is a connection to that path, it is terminated.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `path` contains a handle identifying the I/O path.
  
  This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.

Indirect Errors

- `EOS_BPNUM` Returned when the path number is invalid.
- `EOS_PTHLOST` Returned when the path is lost and no longer valid.

See Also

ite_path_open()
ite_path_dup()
Duplicates Path and Shares Connection

Syntax
#include <SPF/item.h>
error_code ite_path_dup(
    path_id      dup_path,
    path_id      *new_path);

Libraries
item.l

Description
ite_path_dup() creates a new path ID referencing the same path as the original.

ite_path_dup() is similar to ite_path_clone(), except that if the original path has
an active connection, the ite_clone_path() creates a new connection to the same
two endpoints used by the original path. Also, unlike cloned paths, duplicated paths
point to the same SoftStax path description.

With ite_path_dup(), both paths share the same connection. If one path terminates
the connection, both paths lose that connection to the endpoint. However, if the
new path created by this call is closed, the original path’s connection remains open.

Attributes
Operating System:       OS-9 and OS-9 for 68K
State:                  User
Threads:                Safe

Parameters

dup_path             specifies the path to duplicate.
new_path             points to where the new paths’s ID is returned.

Indirect Errors
EOS_BPNUM            Returned when the path number is invalid.
EOS_PTHLOST           Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND       Returned when the driver cannot find its local path storage
                      for the path_id passed in.

See Also
ite_path_clone()
ite_path_open()

Opens a Path

Syntax

#include <SPF/item.h>
error_code ite_path_open(
    char    *dev_name,
    u_int32  mode,
    path_id  *new_path,
    addr_type *our_num);

Libraries

item.l

Description

ite_path_open() performs two operations:

- opens and initializes the device type structure for the path
- initializes the our_num field in the ITEM device’s device_type structure

The result of the ite_path_open() call is that a path to the device pointed to by dev_name is returned. If our_num is not NULL, the call also initializes the dev_ournum structure in the ITEM device. If our_num equals NULL, ITEM uses the default local addressing out of the device descriptor being opened.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

dev_name
points to the string name specifying the device to which you are opening a path.

This string can contain the name of an OS-9 device descriptor that refers to a single protocol driver, a string defining all the protocols to stack on the path, the far end address to convert to, or an OS-9 device descriptor containing the stack to be built.

mode
contains the mode in which to open a device.

Possible bit field values for mode include the following: FAM_READ, FAM_WRITE, or FAM_NONSHARE.
new_path

points to the location where SoftStax returns the new path ID.

our_num

points to the addr_type structure allocated by the application to initialize the dev_ournum field in the ITEM device you are opening. For valid parameters for the addr_type structure, refer to the item.h file.

You can pass our_num as NULL. In this case, the ite_path_open() call uses the default address information for the ITEM device.

Indirect Errors

EOS_EVBSY  Returned when the receive-thread event cannot be created because it already exists.

EOS_MNF  Returned when the device descriptor, driver, or the SoftStax manager is not loaded.

EOS_STKFULL  Returned if more than six protocols are being stacked on the path.

See Also

ite_path_close()

item.h for addr_type parameters
ITE_PATH_POP() Removes Driver from Top of Stack

Syntax

```c
#include <SPF/item.h>
error_code ite_path_pop(path_id);
```

Libraries

item.l

Description

`ite_path_pop()` removes the driver from the top of the protocol stack on the path. It does not alter any other protocols on the stack.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `path` contains a handle identifying the I/O path. This handle is usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.

Indirect Errors

- **EOS_BPNUM**: Returned when the path number is invalid.
- **EOS_BTMSTK**: Returned if the last driver on the stack has already been reached and no more pops can take place.
- **EOS_NOSTACK**: Returned if there are no protocol drivers on the stack.
- **EOS_PTHLOST**: Returned when the path is lost and no longer valid.

See Also

`ite_path_pop()`
ite_path_profileget()  
Get Path Profile

Syntax
#include <SPF/item.h>
#include <SPF/spf_oob.h>

error_code ite_path_profileget(
    path_id      path,
    conn_type    *conn,
    u_int32      *pr_size,
    void         *pr_buffer);

Libraries
item.l

Description
The conn_type pointer may be passed in by the application as NULL or a pointer to a valid conn_type structure. If the conn_type pointer is NULL, the driver should return the per path storage profile structure (pp_profile) in the user buffer. If the conn_type pointer is not NULL, the conn_svc_type field in the conn_type pointer is set to the ITE_SVC_xxx for the profile the user wants to get.

If the service profile is not supported, the driver should return EOS_UNKSVC. If the profile number is invalid, the driver should return EOS_ILLPRM. Once the profile structure element in the profile array of the logical unit has been determined, the driver should check to make sure the user buffer size is big enough to fit the entire profile. If not, the driver should copy as much of the profile as can be put in the buffer and return EOS_BADSIZ error. If the buffer is big enough, the driver should copy the profile into the buffer and return SUCCESS.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
path
contains a handle identifying the I/O path.
This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().
ite_path_profileget()

conn
points to the caller-allocated conn_type structure set up to represent the connection type information for your device.

pr_size
points to the number of bytes in the pr_buffer to read.

pr_buffer
points to the caller-allocated buffer in which the path profile is placed.

Indirect Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_BPNUM</td>
<td>Returned when the path number is invalid.</td>
</tr>
<tr>
<td>EOS_PTHLOST</td>
<td>Returned when the path is lost and no longer valid.</td>
</tr>
<tr>
<td>EOS_PPS_NOTFND</td>
<td>Returned when the driver cannot find its local path storage for the path_id passed in.</td>
</tr>
<tr>
<td>EOS_ILLPATH</td>
<td>Returned when conn_type structure is NULL, or conn_svc_type in conn_type structure is invalid.</td>
</tr>
</tbody>
</table>

Other errors may be returned by SoftStax drivers.

See Also

ite_path_profileset()
ite_path_profileset()

Set Path Profile

Syntax

```
#include <SPF/item.h>
#include <SPF/spf_oob.h>

error_code ite_path_profileset(
    path_id    path,
    conn_type  *conn,
    u_int32    *pr_size,
    void       *pr_buffer);
```

Description

For the set profile setstat, the user passes in the new parameters for the profile of the path. The driver should attempt to validate the new parameters of the profile as needed to ensure the application has not made any illegal parameter settings for the protocol. If so, EOS_ILLPRM can be returned. This profile should be copied into the pp_profile structure in the per path storage by the driver.

If the buffer field is NULL and the parameter field is non-NULL, the user has passed in a valid conn_type pointer. The conn_svc_type field in the conn_type structure should be used by the driver to set the new profile for the path. For instance, when a path opens and the default profile is a voice call, this profile is copied into the per path storage. If a set profile call is made with a conn_type structure and the conn_svc_type field is ITE_SVC_DATA_ANY, the driver should copy the ITE_SVC_DATA_ANY profile from the logical unit array into the per path storage profile structure.

If the buffer field is non-NULL, then it points to an xxx_profile structure that has probably been modified by the application. The driver should do some integrity checking on the modifications, then change the profile structure in the per path storage to the contents of the passed in buffer.

Once copied, this profile becomes valid for the path. Notice the only way to set profiles in the logical unit array is to make a change to the descriptor.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
Parameters

path
contains a handle identifying the I/O path.
This handle is usually obtained from calls such as ite_path_open(),
_os_open(), and socket().

conn
points to the caller-allocated conn_type structure set up to represent the
connection type information for your device.

pr_size
points to the number of bytes to read in the pr_buffer.

pr_buffer
points to the caller-allocated buffer in which the path profile is placed.

Indirect Errors

EOS_BPNUM Returned when the path number is invalid.
EOS_PTHLOST Returned when the path is lost and no longer valid.
EOS_PPS_NOTFND Returned when the driver cannot find its local path storage
for the path_id passed in.
EOS_ILLPATH Returned when conn_type structure is NULL, or
conn_svc_type in conn_type structure is invalid.
Other errors may be returned by SoftStax drivers.

See Also

ite_path_profileget()
ite_path_push()

Pushes Protocol or Hardware-Driver Onto Path

Syntax

```c
#include <SPF/item.h>
error_code ite_path_push(
    path_id    path,
    char       *dev_name);
```

Libraries

item.l

Description

ite_path_push() pushes a protocol or hardware driver onto the path.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

path
    contains a handle identifying the I/O path.
    This handle is usually obtained from calls such as ite_path_open(), _os_open(), and socket().

dev_name
    Contains the ITEM device name string referencing the ITEM device to be stacked on this path.

Indirect Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOS_MNF</td>
<td>Returns when the module referenced by dev_name is not currently in memory.</td>
</tr>
<tr>
<td>EOS_STKFULL</td>
<td>Returned if there are already six protocols stacked on the path being pushed.</td>
</tr>
<tr>
<td>EOS_BUSERR</td>
<td>Returned if dev_name is NULL.</td>
</tr>
<tr>
<td>EOS_BPNUM</td>
<td>Returned when the path number is invalid.</td>
</tr>
<tr>
<td>EOS_PTHLOST</td>
<td>Returned when the path is lost and no longer valid.</td>
</tr>
<tr>
<td>EOS_PPS_NOTFND</td>
<td>Returned when the driver cannot find its local path storage for the path_id passed in.</td>
</tr>
</tbody>
</table>

See Also

ite_path_pop()
Conv and OS Functions

This section contains descriptions of the telecommunications Application Programming Interface (API) functions.

The following table lists and describes the functions that compose the conv and OS libraries.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_getstat()</td>
<td>Getstat</td>
</tr>
<tr>
<td>_os_setstat()</td>
<td>Setstat</td>
</tr>
</tbody>
</table>
### Syntax

```c
#include <sg_codes.h>
#include <types.h>
#include <SPF/spf.h>
error_code _os_getstat(
    path_id      path,
    u_int32      code,
    void         *pb);
```

### Libraries

- `conv_lib.l` (OS-9 for 68K)
- `os_lib.l` (OS-9)

### Description

`_os_getstat()` is a wildcard call used to get individual device parameters that are not uniform on all devices or that are highly hardware dependent.

⚠️ This call is only available for OS-9 for 68K with `conv_lib.l` is used.

### Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

### Parameters

- `path` contains a handle identifying the I/O path, usually obtained from calls such as `ite_path_open()`, `_os_open()`, and `socket()`.
- `code` contains the getstat code.
- `pb` points to the getstat `spf_ss_pb` parameter block.

### Indirect Errors

- `EOS_BPNUM` Returned when the path number is invalid.
- `EOS_PTHLOST` Returned when the path is lost and no longer valid.
- `EOS_PPS_NOTFND` Returned when the driver cannot find its local path storage for the `path_id` passed in.

### See Also

- `_os_setstat()`
_os_setstat()
Setstat

Syntax
#include <sg_codes.h>
#include <types.h>
#include <SPF/spf.h>
error_code _os_setstat(
    path_id      path,
    u_int32      code,
    void         *pb);

Libraries
conv_lib.l (OS-9 for 68K)
oa_lib.l (OS-9)

Description
_os_setstat() is a wildcard call used to set individual device parameters that are
not uniform on all devices or are highly hardware dependent.

⚠️ This call is only available for OS-9 for 68K with conv_lib.l is used.

Attributes
Operating System:         OS-9 and OS-9 for 68K
State:                   User
Threads:                 Safe

Parameters
path
    contains a handle identifying the I/O path. This handle is obtained from calls
    such as ioe_path_open(), _os_open(), and socket().

code
    contains the setstat code.

pb
    points to the setstat parameter block.

See Also
_os_getstat()
Debugging Functions

The `dbg_mod.l` library enables you to incorporate debugging information into drivers you are developing. The library creates a data module in memory into which the driver can write information. The data module can be reviewed for debugging information. This library allows you to debug in real time.

The following table lists and describes the debugging functions.

Table 2-13. Debugging Functions and Descriptions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>debug_4data()</code></td>
<td>Write Four Longs of Debug Data</td>
</tr>
<tr>
<td><code>debug_data()</code></td>
<td>Write a Character String and One Integer</td>
</tr>
<tr>
<td><code>debug_init()</code></td>
<td>Entry Point</td>
</tr>
<tr>
<td><code>debug_string()</code></td>
<td>Write a Character String in the Data Module</td>
</tr>
<tr>
<td><code>debug_timestamp()</code></td>
<td>Write a Time Stamp into the Data Module</td>
</tr>
</tbody>
</table>
debug_4data()
Write Four Longs of Debug Data

Syntax

```c
void debug_4data (  
    debug_stat,  
    u_int32    a,  
    u_int32    b,  
    u_int32    c,  
    u_int32    d);  
```

Description

This function writes four longs of debug data.

Parameters

- `debug_stat`
  points to modules debug_stat structure.

- `a`
  is the first 32-bit data value.

- `b`
  is the second 32-bit data value.

- `c`
  is the third 32-bit data value.

- `d`
  is the fourth 32-bit data value.
**debug_data()**

Write a Character String and One Integer

**Syntax**

```c
void debug_data (
    Dbg_stat      debug_stat,
    u_int32       *sptr,
    u_int32       data);
```

**Description**

This call writes a character string (no larger than 10 characters, not including the NULL byte at the end of the string) and one `u_int32` into the debug module. It is created to use one full line if you dump the debug module in `rombug` or if you are using the `dump` utility.

**Parameters**

- **debug_stat**
  - points to modules `debug_stat` structure.

- ***sptr**
  - points to an ASCII string with a maximum length of 10 bytes. To ensure readability of the debug module, always make the string exactly 10 bytes.

- **data**
  - is a 32-bit data value.

Refer to the *SoftStax Porting Guide* for more information about debug data string conventions.
debug_init()
Entry Point

Syntax

```c
error_code debug_init (  
    u_int32      size,  
    Dbg_stat    *dbg_ptr,  
    char        *mod_name);  
```

Description

This is the entry point where the caller tells the debug library which data module name to use and the data module size requirements.

It is recommended the module name and data module size parameters be kept in a device descriptor so the calling code can get them out and they do not have to be hard-coded anywhere within the module.

Parameters

- `size` contains the size of the data module.
  It must be greater than 64 bytes.

- `*dbg_ptr` points to the pointer defined in static storage.

- `*mod_name` contains the name of the data module.
  It must not be NULL.
**debug_string()**

Write a Character String in the Data Module

**Syntax**

```c
void debug_string (  
   Dbg_stat     debug_stat,  
   u_char       *dptr,  
   u_int32      size);
```

**Description**

This call writes a character string of size bytes into the data module. The function rounds up to the nearest dump line, padding with zeros, so the debug module remains readable when dumped in rombug or when using the dump utility.

**Parameters**

- **debug_stat**
  points to modules debug_stat structure.

- **dptr**
  points to an ASCII string.

- **size**
  contains the size of the string to place into the debug module.
**Syntax**

```c
void debug_timestamp (  
    Dbg_stat   debug_stat,  
    u_int32    x,  
    u_int32    y);
```

**Description**

This function writes a time stamp into the debug data module. The time-stamp has the following layout:

<table>
<thead>
<tr>
<th>Byte: 0</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seconds elapsed since 01/01/1970</td>
<td>System tick rate</td>
<td>Current clock tick</td>
<td>x</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

- `debug_stat` points to modules debug_stat structure.
- `x` is the first 32-bit data value.Unsigned long you can pass in for debugging.
- `y` is the second 32-bit data value. Unsigned long you can pass in for debugging.
mbuf Functions

An mbuf is a common data structure used to store variable-length data blocks. mbufs can be queued, allocated, and deallocated, and are used for compatibility between local area networking packets and wide area networking packets.

The following table lists the mbuf functions provided in mbuflib.l.

### Table 2-14. mbuf functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_adj()</td>
<td>Adjust Pointers and Sizes</td>
</tr>
<tr>
<td>m_cat()</td>
<td>Append mbuf</td>
</tr>
<tr>
<td>m_copy()</td>
<td>Allocate New mbuf and Copy Data to New mbuf</td>
</tr>
<tr>
<td>m_deq()</td>
<td>Remove First mbuf Packet</td>
</tr>
<tr>
<td>m_enq()</td>
<td>Append mbuf to mbuf Queue</td>
</tr>
<tr>
<td>m_flush()</td>
<td>Free Empty mbufs in Packet Chain</td>
</tr>
<tr>
<td>m_free()</td>
<td>Free mbuf</td>
</tr>
<tr>
<td>m_free_m()</td>
<td>Return mbuf to Free Pool</td>
</tr>
<tr>
<td>m_free_p()</td>
<td>Free Packet Chain</td>
</tr>
<tr>
<td>m_free_q()</td>
<td>Free mbuf Queue</td>
</tr>
<tr>
<td>m_get()</td>
<td>Allocate mbuf (Blocking)</td>
</tr>
<tr>
<td>m_getn()</td>
<td>Allocate mbuf (Non-Blocking)</td>
</tr>
<tr>
<td>m_len_p()</td>
<td>Return Number of Data Bytes</td>
</tr>
<tr>
<td>m_move()</td>
<td>Copy Specified Number of Bytes</td>
</tr>
<tr>
<td>m_msize()</td>
<td>Return Number of Bytes in Packet</td>
</tr>
<tr>
<td>m_pad(), M_PAD()</td>
<td>Advance Data Pointer</td>
</tr>
<tr>
<td>M_PREPEND()</td>
<td>Prepend Space to mbuf</td>
</tr>
<tr>
<td>m_ptod()</td>
<td>Return Pointer to First Valid Byte</td>
</tr>
<tr>
<td>m_pullup()</td>
<td>Ensure First mbuf Has Data</td>
</tr>
<tr>
<td>mtod()</td>
<td>Get Pointer to mbuf Data</td>
</tr>
<tr>
<td>m_unpad(), M_UNPAD()</td>
<td>Move Data Pointer</td>
</tr>
</tbody>
</table>

The m_get(), m_getn(), and m_free() mbuf library functions request and free mbuf structures through a user-installed system call. The remaining functions operate only on the mbuf header fields and are contained within the mbuflib.l library.
Adjust Pointers and Sizes

Syntax

```c
#include <mbuf.h>

u_int32 m_adj(
    mbuf         mb,
    u_int32      count);
```

Description

`m_adj()` adjusts pointers and sizes to remove data from the packet. `m_adj()` does not free the mbufs; only the pointers and sizes are adjusted as indicated. `m_adj()` returns the actual count of data removed.

Parameters

- **mb** is a pointer to the packet chain.
- **count** specifies the amount and location of data to remove. If `count` is positive, the data is removed from the beginning of the packet chain. If `count` is negative, the data is removed from the end of the packet chain.

See Also

- `m_pad()`, `M_PAD()`
- `m_unpad()`, `M_UNPAD()`
m_cat()

Append mbuf

Syntax

```c
#include <mbuf.h>
mbuf m_cat(
    mbuf mb,
    mbuf nb);
```

Description

`m_cat()` appends mbuf packet `nb` to the end of packet `mb`. The result combines the two mbufs into a single packet. `m_cat()` returns `mb`, if `mb` is non-NULL. Otherwise, `m_cat()` returns `nb`.

Parameters

- `mb` is a pointer to the packet chain on to which to append `nb`.
- `nb` is a pointer to the packet chain to append to `mb`. 
Allocate New mbuf and Copy Data to New mbuf

Syntax

```c
#include <mbuf.h>

mbuf m_copy(
    mbuf mb,
    u_int32 offset,
    u_int16 count,
    int32 *status);
```

Description

`m_copy()` allocates a new mbuf and copies data beginning from the original mbuf to the new mbuf. `m_copy()` returns NULL if `offset` points past the end of the packet. If less than `count` bytes are in the packet following the `offset`, the returned mbuf is at least of size `count`, but it is not full. `m_copy()` calls `m_getn()` to allocate the new mbuf. If `m_getn()` fails, `m_copy()` returns NULL and places the error code status from `m_getn()` in `status`.

Parameters

- **mb**
  - is a pointer to the original mbuf.
- **offset**
  - specifies the location to begin the copy.
- **count**
  - specifies the number of bytes to copy.
- **status**
  - is a pointer to the returned error status code.

Fatal Errors

- **EOS_IllArg** Returned when the request is for a size greater than 32767 bytes.
- **EOS_MemFull** Returned when system memory is full.
- **ENoBUFs** Returned when a desired mbuf is unavailable.

See Also

`m_getn()`
m_deq()

Remove First mbuf Packet

Syntax

```c
#include <mbuf.h>
mbuf m_deq(mbuf *q);
```

Description

`m_deq()` removes the first mbuf packet chain from an mbuf queue. `m_deq()` returns a pointer to the removed mbuf packet chain.

Parameters

- `q` is a pointer to a pointer to the queue.

See Also

- `m_enq()`
m_enq()

Append mbuf to mbuf Queue

Syntax

#include <mbuf.h>

void m_enq(
    mbuf *q,
    mbuf mb);

Description

m_enq() appends an mbuf packet chain to the end of an mbuf queue.

Parameters

q
    is a pointer to a pointer to the original mbuf queue.

mb
    is a pointer to the mbuf packet.

See Also

m_deq()
Syntax
#include <mbuf.h>
mbuf m_flush(mbuf mb);

Description
m_flush() frees all empty mbufs in the mbuf packet chain mb. m_flush() returns a pointer to the new head of the mbuf packet chain.

Parameters
mb
is a pointer to the mbuf packet chain.

See Also
m_free()
m_free()
Free mbuf

Syntax
#include <mbuf.h>
mbuf m_free(mbuf mb);

Description
m_free() returns the specified mbuf to the free pool. m_free() returns a pointer to the next mbuf in the mbuf packet chain or NULL if this was the last mbuf in the packet chain.

If the SPF_NOFREE bit is set in the M_FLAGS field of the mbuf header then the mbuf is not freed, instead the SPF_DONE bit is set in the m_flag field.

Parameters

mb
is a pointer to the mbuf to free.

See Also
m_free_m()
m_free_p()
m_free_q()
m_free_m()

Return mbuf to Free Pool

Syntax
#include <mbuf.h>
int32 m_free_m(mbuf mb);

Description
m_free_m() returns the specified mbuf to the free pool. m_free_m() returns 0 if successful. Non-zero return values indicate the system memory data structure is corrupted.

If the SPF_NOFREE bit is set in the M_FLAGS field of the mbuf header then the mbuf is not freed, instead the SPF_DONE bit is set in the m_flag field.

Parameters
mb
is a pointer to the mbuf to free.

You must ensure mb is a pointer to an address returned by m_get() or m_getn().

See Also
m_free()
m_free_p()
m_free_q()


Syntax

```c
#include <mbuf.h>
mbuf m_free_p(mbuf mb);
```

Description

`m_free_p()` returns an entire `mbuf` packet chain to the free pool. All `mbuf`s in the packet are freed. `m_free_p()` returns a pointer to the next `mbuf` packet chain in the `mbuf` queue or `NULL` if this was the last packet chain in the queue.

If the `SPF_NOFREE` bit is set in the `M_FLAGS` field of the `mbuf` header then the `mbuf` is not freed, instead the `SPF_DONE` bit is set in the `m_flag` field.

Parameters

- `mb` is a pointer to the `mbuf` packet chain to free.

See Also

- `m_free()`
- `m_free_m()`
- `m_free_q()`
m_free_q()

Free mbuf Queue

Syntax

#include <mbuf.h>

void m_free_q(mbuf *queue);

Description

m_free_q() returns the entire mbuf queue to the free pool. All mbufs for all packet chains in the queue are freed.

If the SPF_NOFREE bit is set in the M_FLAGS field of the mbuf header then the mbuf is not freed, instead the SPF_DONE bit is set in the m_flag field.

Parameters

queue

is a pointer to a pointer to the mbuf queue to free.

See Also

m_free()
m_free_m()
m_free_p()
m_get()

Allocate mbuf (Blocking)

Syntax

```c
#include <mbuf.h>
mbuf m_get(
    u_int16      size,
    int32        *status);
```

Description

Given a size, m_get() allocates an mbuf with a data area of size bytes. m_get() blocks until an mbuf of the requested size is available. If an error occurs, m_get() returns NULL and sets status to the error code. Otherwise, m_get() returns a pointer to the allocated mbuf.

Parameters

- `size` specifies the number of bytes to allocate for the data area. m_get() can allocate a maximum of 0XFFFF minus the size of the mbuf header, which is 0X010.

- `status` is a pointer to the returned error status code.

Non-fatal Errors

- **EOS_IllArg** Returned when the request is for a size greater than 32767 bytes.
- **EOS_MemFull** Returned when no more system memory is available.
- **EnoBufs** Returned when an mbuf of the requested size is unavailable.

See Also

- m_getn()
m_getn()
Allocate mbuf (Non-Blocking)

Syntax
#include <mbuf.h>
mbuf m_getn(
    u_int16   size,
    int32     *status);

Description
m_getn() allocates an mbuf with a data area of size bytes and returns a pointer to
it. If the memory request cannot be granted immediately, m_getn() returns NULL and
ENOBUFFS is returned in status. If an error occurs, m_getn() returns NULL and sets
status to the error code. Otherwise, m_getn() returns a pointer to the allocated
mbuf.

Parameters
size specifies the number of bytes to allocate for the data area. m_getn() can
allocate a maximum of 0XFFFFF minus the size of the mbuf header, which is
0X010.

status is a pointer to the returned error status code.

Non-fatal Errors
EOS_ILLARG Returned when the request is for a size greater than 32767
bytes.
EOS_MEMFUL Returned when no more system memory is available.
ENOBUFFS Returned when an mbuf of the requested size is unavailable.

See Also
m_get()
m_len_p()

Return Number of Data Bytes

Syntax

#include <mbuf.h>
u_int32 m_len_p(mbuf mb);

Description

m_len_p() adds up the total number of bytes used by each mbuf data allocation in the packet list. m_len_p() returns the total number of data bytes in the mbuf packet chain.

Parameters

mb

is a pointer to the packet.
m_move()
Copy Specified Number of Bytes

Syntax

```
#include <mbuf.h>

u_int32 m_move(
    mbuf        mb,
    u_int32     offset,
    u_int32     count,
    char        *buffer);
```

Description

m_move() copies count bytes in the mbuf packet chain starting at offset into buffer. The mbuf remains unchanged. m_move() returns the count of bytes actually copied.

Parameters

mb

is a pointer to the mbuf packet chain.

offset

specifies the number of bytes from the front of the buffer into which to begin copying.

count

specifies the number of bytes to copy.

buffer

is a pointer to the buffer in which to copy the data.
m_msize()
Return Number of Bytes in Packet

Syntax
#include <mbuf.h>

u_int32 m_msize(mbuf mb);

Description
m_msize() adds up the total number of bytes used by the entire mbuf packet chain including headers, data, and non-used memory. m_msize() returns the number of bytes in the specified mbuf packet chain.

Parameters
mb
is a pointer to the mbuf packet chain.

See Also
m_len_p()
m_pad(), M_PAD()

Advance Data Pointer

Syntax

```c
#include <mbuf.h>

u_int32 m_pad(
    mbuf mb,
    u_int16 count);

void M_PAD(
    mbuf mb,
    u_int16 count);
```

Description

`m_pad()` advances the data pointer in the mbuf. `m_pad()` returns 0 if successful and adjusts the data pointer. If the request exceeds the available space, the amount of space available is returned and the data pointer remains unchanged.

A macro, `M_PAD()`, is also available and has the same functionality as `m_pad()`. However, `M_PAD()` does not check to see if the request exceeds the available space.

Parameters

- **mb** is a pointer to the mbuf.
- **count** specifies the number of bytes to advance the data pointer.

See Also

- `m_adj()`
- `m_unpad()`, `M_UNPAD()`
m_ptod()

Return Pointer to First Valid Byte

Syntax
#include <mbuf.h>
char *m_ptod(mbuf mb);

Description
m_ptod() returns a pointer to the first valid data byte in an mbuf packet chain.

Parameters
mb
  is a pointer to the mbuf packet chain.

See Also
mtod()
Syntax

```
#include <mbuf.h>

u_int32 M_PREPEND ( 
    mbuf mbp, 
    u_int16 count, 
    u_int16 how;
```

**Description**

`M_PREPEND()` prepends the number of bytes specified in the front of the data. If there is not enough space, return the NULL mbuf pointer.

**Parameters**

- `mbp` is a pointer to the mbuf.
- `count` specifies the number of bytes to prepend.
- `how` is not currently used.
Syntax

```c
#include <mbuf.h>
int32 m_pullup(
    mbuf *mb,
    u_int16 count);
```

Description

`m_pullup()` ensures the first `mbuf` on the packet chain contains at least `count` bytes of data. An implicit `m_getn()` is performed if necessary. `m_pullup()` returns 0 if successful. The replaced `mbufs` are automatically freed. It returns a non-zero error code if the `m_getn()` failed or the `mbuf` chain did not contain the requested amount of data. In this case, the original `mbuf` packet chain remains untouched.

Parameters

- `mb` is a pointer to a pointer to the first `mbuf`.
- `count` specifies the minimum amount of data for the first `mbuf` to contain.

Non-fatal Errors

- **EOS_Illarg** Returned when the request is for a size greater than 32767 bytes.
- **EBUFTOOSMALL** Returned when `count` is greater than the memory within the `mbuf`.
- **ENOBUFFS** An `mbuf` of the requested size is unavailable.

See Also

- `m_getn()`
mtod()  
Get Pointer to mbuf Data

Syntax

```c
#include <mbuf.h>
(t) *mtod(
    mbuf m,
    typedef t);
```

Description

`mtod()` is a macro returning a pointer to the data area in an mbuf packet. `mtod()` returns a pointer with type `t` which points to the first byte of data in the mbuf packet. This pointer is obtained by adding the `m_offset` field of the mbuf packet to the beginning address of the mbuf.

Parameters

- `m` is a pointer to the mbuf.
- `t` is the type with which to cast the returned pointer.

See Also

`m_ptod()`
m_unpad(), M_UNPAD()  
Move Data Pointer

Syntax

```c
#include <mbuf.h>
u_int32 m_unpad(
    mbuf mb,
    u_int16 count);

void M_UNPAD(
    mbuf mb,
    u_int16 count);
```

Description

`m_unpad()` tries to move the data pointer so there are `count` bytes at the front of the mbuf. If successful, `m_unpad()` returns 0 and adjusts the data pointer. If the request exceeds the available space, the amount of space available is returned and the data pointer remains unchanged.

A macro, `M_UNPAD()`, is also available and has the same functionality as `m_unpad()`. However, `M_UNPAD()` does not check to see if the request exceeds the available space.

Parameters

- `mb` is a pointer to the mbuf.
- `count` specifies the number of bytes to leave before the data pointer.

See Also

- `m_adj()`
- `m_pad()`, `M_PAD()`
RPC Functions

The RPC library routines enable C programs to make procedure calls on machines across a network.

The following table lists and describes the RPC C library functions.

**Table 2-15. RPC C Library Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_destroy()</td>
<td>Destroy Authentication Information</td>
</tr>
<tr>
<td>authnone_create()</td>
<td>Create Authentication Handle</td>
</tr>
<tr>
<td>authunix_create()</td>
<td>Create Authentication Handle</td>
</tr>
<tr>
<td>authunix_create_default()</td>
<td>Create Authentication Handle</td>
</tr>
<tr>
<td>callrpc()</td>
<td>Call a Remote Procedure</td>
</tr>
<tr>
<td>clnt_broadcast()</td>
<td>Broadcast an RPC Call</td>
</tr>
<tr>
<td>clnt_call()</td>
<td>Call Remote Procedure</td>
</tr>
<tr>
<td>clnt_control()</td>
<td>Change Client Object</td>
</tr>
<tr>
<td>clnt_create()</td>
<td>Create Client Handle</td>
</tr>
<tr>
<td>clnt_destroy()</td>
<td>Destroy Client Handle</td>
</tr>
<tr>
<td>clnt_freeres()</td>
<td>Free Data Area Associated with Result</td>
</tr>
<tr>
<td>clnt_geterr()</td>
<td>Copy Client Error Structure</td>
</tr>
<tr>
<td>clnt_pcreateerror()</td>
<td>Print Message to Standard Error</td>
</tr>
<tr>
<td>clnt_perrno()</td>
<td>Print Message to Standard Error</td>
</tr>
<tr>
<td>clnt_perror()</td>
<td>Print Message to Standard Error</td>
</tr>
<tr>
<td>clnt_spcreateerror()</td>
<td>Encode Message to a Buffer</td>
</tr>
<tr>
<td>clnt_sperrno()</td>
<td>Encode Message to a Buffer</td>
</tr>
<tr>
<td>clnt_sperror()</td>
<td>Encode Message to a Buffer</td>
</tr>
<tr>
<td>clntraw_create()</td>
<td>Create Loopback Client Handle</td>
</tr>
<tr>
<td>clnttcp_create()</td>
<td>Create Client Handle Using TCP</td>
</tr>
<tr>
<td>clntudp_create()</td>
<td>Create Client Handle Using UDP</td>
</tr>
<tr>
<td>get_myaddress()</td>
<td>Return Local Machine’s Internet Address</td>
</tr>
<tr>
<td>pmap_getmaps()</td>
<td>Return List of Program to Port Mappings</td>
</tr>
<tr>
<td>pmap_getport()</td>
<td>Return Service Port Number</td>
</tr>
<tr>
<td>pmap_rmtcall()</td>
<td>Request Portmap to Make an RPC Call</td>
</tr>
<tr>
<td>pmap_set()</td>
<td>Establish Mapping for RPC Service</td>
</tr>
<tr>
<td>pmap_unset()</td>
<td>Destroy Mapping for RPC Service</td>
</tr>
<tr>
<td>registerrpc()</td>
<td>Register RPC Service with Portmap</td>
</tr>
<tr>
<td>svc_destroy()</td>
<td>Destroy Service Transport Handle</td>
</tr>
<tr>
<td>svc_freeargs()</td>
<td>Free Data Area for Parameters</td>
</tr>
<tr>
<td>svc_getargs()</td>
<td>Decode Parameters of a Service Request</td>
</tr>
<tr>
<td>svc_getcaller()</td>
<td>Get Network Address of Caller</td>
</tr>
<tr>
<td>svc_getreq()</td>
<td>Custom Asynchronous Event Processor</td>
</tr>
<tr>
<td>svc_getreqset()</td>
<td>Custom Asynchronous Event Processor</td>
</tr>
<tr>
<td>svc_register()</td>
<td>Establish Service with Dispatch Routine</td>
</tr>
</tbody>
</table>
### Table 2-15. RPC C Library Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>svc_run()</td>
<td>Process RPC Request</td>
</tr>
<tr>
<td>svc_sendreply()</td>
<td>Send RPC Results</td>
</tr>
<tr>
<td>svc_unregister()</td>
<td>Remove Mapping for RPC Service</td>
</tr>
<tr>
<td>svc_err_auth()</td>
<td>Report Authentication Error</td>
</tr>
<tr>
<td>svc_err_decode()</td>
<td>Report Decoding Error</td>
</tr>
<tr>
<td>svc_err_noproc()</td>
<td>Report Unknown Procedure Number</td>
</tr>
<tr>
<td>svc_err_noprog()</td>
<td>Report Unknown Program Number</td>
</tr>
<tr>
<td>svc_err_progvers()</td>
<td>Report Unknown Version Number</td>
</tr>
<tr>
<td>svc_err_systemerr()</td>
<td>Report System Error</td>
</tr>
<tr>
<td>svc_err_weakauth()</td>
<td>Report Weak Authentication</td>
</tr>
<tr>
<td>svcfd_create()</td>
<td>Create Service Transport on Open Descriptor</td>
</tr>
<tr>
<td>svcraw_create()</td>
<td>Create Loopback Service Transport</td>
</tr>
<tr>
<td>svctcp_create()</td>
<td>Create TCP Service Transport</td>
</tr>
<tr>
<td>svcudp_create()</td>
<td>Create UDP Service Transport</td>
</tr>
</tbody>
</table>
**Syntax**

```c
#include <RPC/rpc.h>
void auth_destroy(AUTH *auth)
```

**Description**

The `auth_destroy()` macro destroys the authentication information associated with `auth`. Destruction usually involves deallocation of private data structures.

**Attributes**

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User
- **Threads**: Safe

**Parameters**

- `auth` is a pointer to the authentication handle.
  
  The use of `auth` is undefined after calling `auth_destroy()`.
authnone_create()
Create Authentication Handle

Syntax
#include <RPC/rpc.h>

AUTH * authnone_create(void)

Description

authnone_create() creates and returns an authentication handle that passes nonusable authentication information with each remote procedure call. This is the default authentication used by RPC.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
authunix_create()
Create Authentication Handle

Syntax

```
#include <RPC/rpc.h>

AUTH * authunix_create(
    char    *machname,
    int     uid,
    int     gid,
    int     len,
    int     aup_gids)
```

Description

authunix_create() creates and returns an authentication handle that contains authentication information. The len and aup_gids parameters are ignored.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- **machname**
  - is the name of the machine the information was created on.
- **uid**
  - is the user’s OS-9 user ID.
- **gid**
  - is the user’s OS-9 group ID.
- **len**
  - is the number of elements in aup_gids.
- **aup_gids**
  - is a reference to a counted array of user’s groups.
authunix_create_default()

Create Authentication Handle

Syntax

```c
#include <RPC/rpc.h>
AUTH * authunix_create_default(void)
```

Description

`authunix_create_default()` calls `authunix_create()` using the appropriate parameters.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
callrpc()

Call a Remote Procedure

Syntax

#include <RPC/rpc.h>

int callrpc(
    char           *host,
    int            prognum,
    int            versnum,
    int            procnum,
    xdrproc_t      inproc,
    char           *in,
    xdrproc_t      outproc,
    char           *out)

Description

callrpc() calls the remote procedure associated with prognum, versnum, and procnum on the machine host. callrpc() returns a value of 0 if it succeeds or the value of the error cast to an integer if it fails. This routine is useful for translating failure statuses into messages.

Attributes

Operating System:  OS-9 and OS-9 for 68K
State:  User
Threads:  Safe

Parameters

host
    is the name of the machine the info was created on.

prognum
    is the program number.

versnum
    is the version number.

procnum
    is the procedure number.

inproc
    codes the parameters.

in
    is the address of the parameter(s).
outproc
decodes the results.

out is the address of where to place the result(s).

⚠️ Calling remote procedures with this routine uses TCP as a transport. You do not have control over time-outs or authentication when using this routine.
clnt_broadcast()

Broadcast an RPC Call

Syntax

```c
#include <RPC/rpc.h>
enum clnt_stat clnt_broadcast(
    u_long           prog,
    u_long           vers,
    u_long           proc,
    xdrproc_t        xargs,
    caddr_t          argsp,
    xdrproc_t        xresults,
    caddr_t          resultsp,
    resultproc_t     eachresult)
```

Description

clnt_broadcast() calls a remote procedure by broadcasting the call message to all locally connected systems.

If eachresult() returns 0, clnt_broadcast() waits for more replies. Otherwise, it returns with the appropriate status.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `prog` is the program number.
- `vers` is the version number.
- `proc` is the procedure number.
- `xargs` is the address of the parameter(s).
- `argsp` is the address of where to place the result(s).
xresultsen
codes the parameters.

resultsp
decodes the results.

eachresult
is called each time clnt_broadcast() receives a response.

Broadcast packets are limited in size to the maximum transfer unit of the data link. For ethernet, this value is 1500 bytes.
clnt_call()

Syntax

```c
#include <RPC/rpc.h>
enum clnt_stat clnt_call(
    CLIENT             *rh,
    u_long             proc,
    xdrproc_t          xargs,
    caddr_t            argsp,
    xdrproc_t          xres,
    caddr_t            resp,
    struct timeval     timeout)
```

Description

The `clnt_call()` macro calls the remote procedure associated with the client handle. The client handle is obtained from a client creation routine such as `clnt_create()`.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `rh` is the client handle.
- `proc` is the procedure number.
- `xargs` encodes the parameters.
- `argsp` decodes the results.
- `xres` is the address of the parameter(s).
- `resp` is the address of where to place the result(s).
- `timeout` is the time allowed for the results to return.
Syntax

```c
#include <RPC/rpc.h>
bool_t clnt_control(
    CLIENT     *cl,
    int        request,
    char       *info)
```

Description

The `clnt_control()` macro changes or retrieves information concerning a client object. These include the parameters timeout, retry count, and server address. The requests are shown in Table 2-16.

<table>
<thead>
<tr>
<th>Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLSET_TIMEOUT</td>
<td>Set timeout value.</td>
</tr>
<tr>
<td>CLGET_TIMEOUT</td>
<td>Get timeout value.</td>
</tr>
<tr>
<td>CLSET_RETRY_TIMEOUT</td>
<td>Set retry timeout value.</td>
</tr>
<tr>
<td>CLGET_RETRY_TIMEOUT</td>
<td>Get retry timeout value.</td>
</tr>
<tr>
<td>CLGET_SERVER_ADDR</td>
<td>Get server <code>inet addr</code>.</td>
</tr>
</tbody>
</table>

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User

Threads: Safe

Parameters

c is the client handle.

request indicates the type of operation.

info is a pointer to the information.
clnt_create()
Create Client Handle

Syntax
#include <RPC/rpc.h>
CLIENT * clnt_create (  
    char         *hostname,  
    unsigned     prog,  
    unsigned     vers,  
    char         *proto)

Description
clnt_create() is a generic client creation routine. Default time outs are set, but you can modify them using clnt_call() or clnt_control().

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
hostname
identifies the remote host.

prog
is the program number.

vers
is the version number.

proto
is the transport protocol to be used.

The currently supported values for proto are tcp and udp.

Using clnt_create() with UDP has its shortcomings. Because messages can only hold up to 8K of encoded data, you cannot use this transport for procedures that take large parameters or return large results.
clnt_destroy()  
Destroy Client Handle

Syntax

#include <RPC/rpc.h>

void clnt_destroy(CLIENT *rh)

Description

The `clnt_destroy()` macro destroys the client’s handle. Destruction usually involves deallocation of private data structures, including its own structure. If the library opened the associated socket, it will close the socket. Otherwise, the socket remains open.

Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

Parameters

`rh`  
is a pointer to the client handle.  
Use of `rh` is undefined after calling `clnt_destroy()`.
clnt_freeres()
Free Data Area Associated with Result

Syntax
#include <RPC/rpc.h>
bool_t clnt_freeres(
    CLIENT    *rh,
    xdrproc_t xres,
    char      *resp);

Description
The clnt_freeres() macro frees any data allocated by the system when the results
of a call were decoded. If successful, this routine returns a value of 1. Otherwise, it
returns a value of 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
rh
    is the client handle.

xres
    is the routine describing results in simple primitive.

resp
    is the address of the results.
Syntax
#include <RPC/rpc.h>
void clnt_geterr(
    CLIENT     *rh,
    struct rpc_err   *errp);

Description
The clnt_geterr() macro copies the error structure out of the client handle to the structure at address errp.

Attributes
 Operating System: OS-9 and OS-9 for 68K
 State: User
 Threads: Safe

Parameters
 rh  
 is a pointer to the client handle.

 errp  
 is a pointer to rpc_err structure.
clnt_pcreateerror()
Print Message to Standard Error

Syntax
#include <RPC/rpc.h>
void clnt_pcreateerror(char *msg)

Description
clnt_pcreateerror() prints a message to standard error indicating why a client handle could not be created. The message is prepended with string msg and a colon (:) clnt_pcreateerror() is used when a clnt_create() call fails.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
msg
is a string to print.
clnt_perrno()
Print Message to Standard Error

Syntax
#include <RPC/rpc.h>
void clnt_perrno(enum clnt_stat num)

Description
clnt_perrno() prints a message to standard error corresponding to the condition indicated by num.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
num
    specifies client error status.
clnt_perror()
Print Message to Standard Error

Syntax
#include <RPC/rpc.h>
void clnt_perror(
    CLIENT     *clnt,
    char       *msg)

Description
clnt_perror() prints a message to standard error indicating why a call failed. clnt is the handle used to perform the call. The message is prepended with string msg and a colon (:).

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User
Threads:            Safe

Parameters
clnt
    is a client handle.
msg
    is a message to print.
clnt_spcreateerror()

Encode Message to a Buffer

Syntax

#include <RPC/rpc.h>

char * clnt_spcreateerror(char *msg)

Description

clnt_spcreateerror() operates like clnt_pcreateerror() except that it returns a pointer to a string instead of printing to standard error.

clnt_spcreateerror() returns a pointer to static data. This static data area is overwritten on each call.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

msg

is a message to encode.
clnt_sperrno()
Encode Message to a Buffer

Syntax
#include <RPC/rpc.h>
char * clnt_sperrno(enum clnt_stat num)

Description
clnt_sperrno() accepts the same parameters as clnt_perrno(). However, instead of sending a message to standard error indicating why a call failed, it returns a pointer to a string containing the message. The string ends with a newline character.

clnt_sperrno() returns a pointer to static data. This static data area is overwritten on each call.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User
Threads:            Safe

Parameters
num
is the client error status.
Syntax

```c
#include <RPC/rpc.h>

char * clnt_sperror(
    CLIENT      *clnt,
    char        *msg)
```

Description

`clnt_sperror()` is similar to `clnt_perror()`. However, `clnt_sperror()` returns a pointer to a string instead of printing to standard error.

`clnt_sperror()` returns a pointer to static data. This static data area is overwritten on each call.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `clnt` is the client handle.
- `msg` is the message to encode.
**clntraw_create()**

Create Loopback Client Handle

**Syntax**

```
#include <RPC/rpc.h>
CLIENT * clntraw_create(
    u_long     prog,
    u_long     vers)
```

**Description**

`clntraw_create()` creates a local client for the remote program `prog` and version `vers`. The transport used to pass messages to the service is actually a buffer within the process’s address space. Therefore, the corresponding server should be located in the same address space. This allows simulation and acquisition of overheads, such as round trip times.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `prog`  
  is the program number.

- `vers`  
  is the version number.
clnttcp_create()
Create Client Handle Using TCP

Syntax
#include <RPC/rpc.h>
CLIENT * clnttcp_create(
    struct sockaddr_in     *addr,
    u_long                 prog,
    u_long                 vers,
    int                    *sockp,
    u_int                  sendsz,
    u_int                  recvsz)

Description
clnttcp_create() creates a client for the remote program prog and version vers. The client uses TCP as a transport.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
addr
    is the internet address of the remote program.
    If addr->sin_port is zero, it is set to the actual port on which the remote program is listening. The remote service is consulted for this information.

prog
    is the program number.

vers
    is the version number.

sockp
    is a pointer to a socket.
    If sockp is null, this routine opens a new socket.

sendsz
    is the size of the send buffer.
    If 0, the default is chosen.

recvsz
    is the size of the receive buffer.
    If 0, the default is chosen.
clntudp_create()

Create Client Handle Using UDP

Syntax

```c
#include <RPC/rpc.h>

CLIENT * clntudp_create(
    struct sockaddr_in     *addr,
    u_long                 prog,
    u_long                 vers,
    struct timeval         wait,
    int                    *sockp)
```

Description

This function creates a client handle for the remote program `prog` and version `vers`. The client uses UDP as a transport. `clntudp_create()` sends the call message until it receives a response or the call times out.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `addr` is the internet address of the remote program.
  If `addr->sin_port` is zero, it is set to the actual port on which the remote program is listening. The remote service is consulted for this information.

- `prog` is the program number.

- `vers` is the version number.

- `wait` is a time-out value used for this call.

- `sockp` is a pointer to a socket. If `sockp` is null, this routine opens a new socket.
get_myaddress()

Return Local Machine’s Internet Address

Syntax
#include <RPC/rpc.h>
void get_myaddress(struct sockaddr_in *addr)

Description
get_myaddress() returns the machine’s address in addr.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
addr
    is the location at which to store the machine’s socket address.
**pmap_getmaps()**

Return List of Program to Port Mappings

**Syntax**

```
#include <RPC/rpc.h>
struct pmaplist *pmap_getmaps(
    struct sockaddr_in *addr)
```

**Description**

`pmap_getmaps()` returns a list of the current program-to-port mappings on the host located at address `addr`. This routine can return null. The `rpcinfo` command uses this routine.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

`addr`

specifies the socket address of the host.
pmap_getport()
Return Service Port Number

Syntax

```c
#include <RPC/rpc.h>

u_short pmap_getport(
    struct sockaddr_in   *addr,
    u_long                prog,
    u_long                vers,
    u_int                 protocol)
```

Description

pmap_getport() returns the port number on which waits a service that supports program `prog` and version `vers` and speaks the transport protocol `protocol`. A return value of 0 indicates that the mapping does not exist or that the system failed to contact the remote portmap service.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `addr` is the socket address.
- `prog` is the program number requested.
- `vers` is the version number requested.
- `protocol` is the protocol to be used.

Supported values are `IPPROTO_UDP` and `IPPROTO_TCP`.

**pmap_rmtcall()**

Request Portmap to Make an RPC Call

**Syntax**

```c
#include <RPC/rpc.h>

enum clnt_stat pmap_rmtcall(
    struct sockaddr_in     *addr,
    u_long                 prog,
    u_long                 vers,
    u_long                 proc,
    xdrproc_t              xdrargs,
    caddr_t                argsp,
    xdrproc_t              xdrves,
    caddr_t                resp,
    struct timeval         tout,
    u_long                 *port_ptr)
```

**Description**

`pmap_rmtcall()` instructs the port mapper on the host at address `addr` to make an RPC call for the user to a procedure on that host.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- **addr**
  - is the socket address.

- **prog**
  - is the program number.

- **vers**
  - is the version number.

- **proc**
  - is the procedure number.

- **xdrargs**
  - is the address of the parameter(s).
argsp
  is the address of where to place the result(s).

xdrres
  encodes the parameters.

resp
  decodes the results.

tout
  is the time allowed for the results to return.

port_ptr
  is the port number.

  port_ptr is modified to the program’s port number if the procedure succeeds.
Syntax

```c
#include <RPC/rpc.h>
bool_t pmap_set(
    u_long      prog,
    u_long      vers,
    int         protocol,
    u_short     port)
```

Description

`pmap_set()` establishes a mapping between `port`, `prog`, `vers`, and `protocol` on the machine's `portmap` service. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `prog` is the program number.
- `vers` is the version number.
- `protocol` is the protocol to be used.
  
  Supported values are `IPPROTO_UDP` and `IPPROTO_TCP`.
- `port` is the port number to associate with the program, version, and protocol triple.
pmap_unset()
Destroy Mapping for RPC Service

Syntax
#include <RPC/rpc.h>
bool_t pmap_unset(
    u_long prog,
    u_long vers)

Description
pmap_unset() destroys all mappings involving program prog and version vers on
the machine’s portmap service. If successful, this routine returns a value of 1.
Otherwise, it returns a value of 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
prog
    is the program number.

vers
    is the version number.
Syntax

```c
#include <RPC/rpc.h>

int registerrpc(
    u_long        prog,
    u_long        vers,
    u_long        proc,
    char          *(*procname)(),
    xdrproc_t     inproc,
    xdrproc_t     outproc)
```

Description

`registerrpc()` registers a procedure with the service package. If a request arrives for program `prog`, version `vers`, and procedure `proc`, `procname` is called with a pointer to its parameter(s). `procname` should return a pointer to its static result(s). This routine returns a value of 0 if the registration succeeded. Otherwise, it returns a value of -1.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `prog` is the program number.
- `vers` is the version number.
- `proc` is the procedure number.
- `procname` is the name of a procedure.
- `inproc` decodes the parameters.
- `outproc` encodes the results.
**Syntax**

```c
#include <RPC/rpc.h>
void svc_destroy(SVCXPRT *xprt)
```

**Description**

`svc_destroy()` destroys the service transport handle `xprt`. Destruction usually involves deallocation of private data structures. This includes its own data structure.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `xprt` is the service transport handle.

Use of `xprt` is undefined after calling this routine.
svc_freeargs()

Free Data Area for Parameters

Syntax

```c
#include <RPC/rpc.h>
bool_t svc_freeargs(
    SVCXPRT *xprt,
    xdrproc_t xargs,
    char *argsp)
```

Description

svc_freeargs() frees any data allocated by the system when the parameters to a service procedure using `xargs` were decoded. This routine returns a value of 1 if the results were successfully freed. Otherwise, it returns a value of 0.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xprt` is the service transport handle.
- `xargs` decodes the parameters.
- `argsp` is the address at which parameters are placed.
svc_getargs()
Decide Parameters of a Service Request

Syntax

```
#include <RPC/rpc.h>
bool_t svc_getargs(
    SVCXPRT       *xprt,
    xdrproc_t     xargs,
    char          *argsp)
```

Description

`svc_getargs()` decodes the parameters of a request associated with `xprt`. This routine returns a value of 1 if decoding succeeds. Otherwise, it returns a value of 0.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xprt` is the service transport handle.
- `xargs` decodes the parameters.
- `argsp` is the address at which parameters are placed.
**svc_getcaller()**

Get Network Address of Caller

**Syntax**

```c
#include <RPC/rpc.h>
struct sockaddr_in *svc_getcaller(SVCXPRT *xprt)
```

**Description**

`svc_getcaller()` gets the network address of the caller of a procedure associated with `xprt`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `xprt` is the service transport handle.
**svc_getreq()**

Custom Asynchronous Event Processor

---

**Syntax**

```c
#include <RPC/rpc.h>
void svc_getreq(int *rdfs);
```

**Description**

`svc_getreq()` is similar to `svc_getreqset()` but limited to 32 descriptors. `svc_run()` makes this interface obsolete.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `rdfs` is the resultant read file descriptor bit mask.
**svc_getreqset()**

Custom Asynchronous Event Processor

**Syntax**

```c
#include <RPC/rpc.h>
void svc_getreqset(int *readfds)
```

**Description**

`svc_getreqset()` is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. `svc_getreqset()` is called when the system has determined that a request has arrived on some socket(s). The routine returns when all sockets associated with the value of `readfds` have been serviced.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `readfds` is the resultant read file descriptor bit mask.
svc_register()

Establish Service with Dispatch Routine

Syntax

```c
#include <RPC/rpc.h>
bool_t svc_register(
    SVCXPRT xprt,
    u_long prog,
    u_long vers,
    void (*dispatch)(,)
    int protocol)
```

Description

`svc_register()` associates `prog` and `vers` with the service dispatch procedure. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xprt` is the service transport handle.
- `prog` is the program number.
- `vers` is the version number.
- `dispatch` is the service dispatch procedure.
- `protocol` is the protocol to be used.

Supported values are `IPPROTO_UDP` and `IPPROTO_TCP`.

The dispatch procedure has the following form (`xprt` is the service transport handle):

```
dispatch(request, xprt)
struct svc_req*request;
```
svc_run()

Process RPC Request

Syntax

```c
#include <RPC/rpc.h>
void svc_run(void)
```

Description

`svc_run()` never returns. It waits for requests to arrive. When a request arrives, `svc_run()` calls the appropriate service procedure. This procedure is usually waiting for I/O.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
svc_sendreply()
Send RPC Results

Syntax

```c
#include <RPC/rpc.h>
bool_t svc_sendreply(
    SVCXPRT   *xprt,
    xdrproc_t xdr_results,
    caddr_t   xdr_location)
```

Description

`svc_sendreply()` is called by a service dispatch routine to send the results of a remote procedure call. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `xprt` is the service transport handle.
- `xdr_results` is the routine used to encode the results.
- `xdr_location` is the address of the results.
svc_unregister()

Remove Mapping for RPC Service

Syntax

```
#include <RPC/rpc.h>
void svc_unregister(
    u_long prog,
    u_long vers)
```

Description

svc_unregister() removes all mapping of the service to dispatch routines and portmap.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- **prog**
  - specifies a remote program.
- **vers**
  - specifies the version of a remote program.
svcerr_auth()  
Report Authentication Error

Syntax
#include <RPC/rpc.h>
void svcerr_auth(
    SVCXPRT *xprt,
    enum auth_stat *why)

Description
svcerr_auth() reports an authentication error. It is called by a service dispatch
routine that refuses to perform a remote procedure call due to an authentication
error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xprt
    is the service transport handle.

why
    is the authentication error status.
svcerr_decode()

Report Decoding Error

Syntax

#include <RPC/rpc.h>

void svcerr_decode(SVCXPRT *xprt)

Description

svcerr_decode() reports a decoding error. It is called by a service dispatch routine that cannot successfully decode its parameters.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

xprt

is the service transport handle.
svcerr_noproc()
Report Unknown Procedure Number

Syntax

```c
#include <RPC/rpc.h>
void svcerr_noproc(SVCXPRT *xprt)
```

Description

svcerr_noproc() reports an unknown procedure number. It is called by a service
dispatch routine that does not implement the procedure number that the caller
requests.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xprt` is the service transport handle.
svcerr_noprog()

Report Unknown Program Number

Syntax

```c
#include <RPC/rpc.h>
void svcerr_noprog(SVCXPRT *xprt)
```

Description

`svcerr_noprog()` reports an unknown program number. It is called when the desired program is not registered with the package. Service implementors usually do not need to use this routine.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xprt` is the service transport handle.
svcerr_progvers()
Report Unknown Version Number

Syntax
#include <RPC/rpc.h>

void svcerr_progvers(SVCXPRT *xprt)

Description

svcerr_progvers() reports an unknown version number. It is called when the desired version of a program is not registered with the package. Service implementors usually do not need to use this routine.

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User
Threads:            Safe

Parameters

xprt

is the service transport handle.
Syntax
#include <RPC/rpc.h>
void svcerr_systemerr(SVCXPRT *xprt)

Description
svcerr_systemerr() reports a system error. It is called by a service dispatch routine when the routine detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xprt
is the service transport handle.
svcerr_weakauth()  
Report Weak Authentication

Syntax

#include <RPC/rpc.h>

void svcerr_weakauth(SVCXPRT *xprt)

Description

svcerr_weakauth() reports weak authentication. It is called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient (but correct) authentication parameters.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

xprt

is the service transport handle.
svcfd_create()
Create Service Transport on Open Descriptor

Syntax
#include <RPC/rpc.h>
SVCXPRT *svcfd_create(
    int fd,
    u_int sendsize,
    u_int recvsize)

Description
svcfd_create() creates a service on top of an open path. Typically, this path is a connected socket for a stream protocol such as TCP.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User
Threads:            Safe

Parameters
fd
    is the path number.
sendsize
    is the size of the send buffer.
    If 0, defaults are chosen.
recvsize
    is the size of the receive buffer.
    If 0, defaults are chosen.
svcraw_create()

Create Loopback Service Transport

Syntax

```c
#include <RPC/rpc.h>
SVCXPRT * svcraw_create(void)
```

Description

`svcraw_create()` creates a local service transport. This routine returns a pointer to the transport. The transport is a buffer within the process’s address space. The corresponding client should live in the same address space. This routine allows for the simulation and acquisition of overheads, such as round trip times.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
svtcp_create()

Create TCP Service Transport

Syntax

```c
#include <RPC/rpc.h>
SVCXprt * svtcp_create(
    int sock,
    u_int sendsize,
    u_int recvsize)
```

Description

svtcp_create() creates a service transport and returns a pointer to it. The transport is associated with the socket `sock`. If the socket is not bound to a local port, this routine binds it to an arbitrary port. This routine chooses suitable defaults if a value of 0 is specified for `sendsize` and `recvsize`.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `sock` is the transport’s socket number.
  - `sock` may be null. In this case, a new socket is created.
- `sendsize` is the size of the send buffer.
- `recvsize` is the size of the receive buffer.
svcudp_create()
Create UDP Service Transport

Syntax
#include <RPC/rpc.h>
SVCXPRT * svcudp_create(int sock)

Description
svcudp_create() creates a service transport and returns a pointer to it. The transport is associated with the socket sock.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
sock
is the transport’s socket number.

sock may be null. In this case, a new socket is created. If the socket is not bound to a local port, this routine binds it to an arbitrary port.
XDR Functions

The XDR C library routines enable C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls are transmitted using these routines.

XDR functions can be found in the rpc.1 library.

The main include file for XDR is RPC/rpc.h, which includes RPC/xdr.h. It can be found in MWOS/SRC/DEFS/SPF/RPC. RPC/xdr.h defines the preliminary XDR structure:

```c
/*
* The XDR handle.
* Contains operation which is being applied to the stream, an
* operations vector for the particular implementation (e.g. see
* xdr_mem.c), and two private fields for the use of the particular
* implementation.
*/
typedef struct {
    enum xdr_opx_op; /* operation; fast additional param */
    struct xdr_ops {
        bool_t(*x_getlong)(); /* get long from underlying stream */
        bool_t(*x_putlong)(); /* put a long to */
        bool_t(*x_getbytes)(); /* get some bytes from */
        bool_t(*x_putchar)(); /* put some bytes to */
        u_int(*x_getpostn)(); /* returns bytes off from beginning */
        bool_t(*x_setpostn)(); /* lets you reposition the stream */
        long *(*x_inline)(); /* buf quick ptr to buffered data */
        void(*x_destroy)(); /* free privates of this xdr_stream */
    } *x_ops;
    caddr_t public; /* users' data */
    caddr_t private; /* pointer to private data */
    caddr_t base; /* private used for position info */
    int handy; /* extra private word */
} XDR;
```

RPCGEN is the compiler that generates client and server sides of an RPC program. The following table lists and describes the XDR C library functions.
Table 2-17. XDR C Library Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdr_accepted_reply()</td>
<td>Encode RPC Reply Messages</td>
</tr>
<tr>
<td>xdr_array()</td>
<td>Translate Arrays to/from XDR</td>
</tr>
<tr>
<td>xdr_authunix_parms()</td>
<td>Generate UNIX Credentials</td>
</tr>
<tr>
<td>xdr_bool()</td>
<td>Translate Booleans to/from XDR</td>
</tr>
<tr>
<td>xdr_bytes()</td>
<td>Translate Counted Bytes to/from XDR</td>
</tr>
<tr>
<td>xdr_callhdr()</td>
<td>Encode RPC Call Header</td>
</tr>
<tr>
<td>xdr_callmsg()</td>
<td>Encode RPC Call Message</td>
</tr>
<tr>
<td>xdr_char()</td>
<td>Translate Characters to/from XDR</td>
</tr>
<tr>
<td>xdr_destroy()</td>
<td>Destroy XDR Stream</td>
</tr>
<tr>
<td>xdr_double()</td>
<td>Translate Double Precision Numbers to/from XDR</td>
</tr>
<tr>
<td>xdr_enum()</td>
<td>Translate Enumerated Types to/from XDR</td>
</tr>
<tr>
<td>xdr_float()</td>
<td>Translate Floating Point Numbers to/from XDR</td>
</tr>
<tr>
<td>xdr_free()</td>
<td>Free XDR Structure</td>
</tr>
<tr>
<td>xdr_getpos()</td>
<td>Return Position in an XDR Stream</td>
</tr>
<tr>
<td>xdr_inline()</td>
<td>Invoke Inline XDR Function</td>
</tr>
<tr>
<td>xdr_int()</td>
<td>Translate Integers to/from XDR</td>
</tr>
<tr>
<td>xdr_long()</td>
<td>Translate Long Integers to/from XDR</td>
</tr>
<tr>
<td>xdrOpaque()</td>
<td>Translate Opaque Data to/from XDR</td>
</tr>
<tr>
<td>xdrOpaque_auth()</td>
<td>Describe RPC Authentication Information</td>
</tr>
<tr>
<td>xdr_pmap()</td>
<td>Describe Procedure Parameters and Port Maps</td>
</tr>
<tr>
<td>xdr_pmaplist()</td>
<td>Describe Procedure Parameters and Port Maps</td>
</tr>
<tr>
<td>xdr_pointer()</td>
<td>Translate Pointer to/from XDR</td>
</tr>
<tr>
<td>xdr_reference()</td>
<td>Translate Pointers to/from XDR</td>
</tr>
<tr>
<td>xdr_rejected_reply()</td>
<td>Encode Rejected RPC Message</td>
</tr>
<tr>
<td>xdr_replymsg()</td>
<td>Encode RPC Reply Message</td>
</tr>
<tr>
<td>xdr_setpos()</td>
<td>Set Position within XDR Stream</td>
</tr>
<tr>
<td>xdr_short()</td>
<td>Translate Short Integers to/from XDR</td>
</tr>
<tr>
<td>xdr_string()</td>
<td>Translate Strings to/from XDR</td>
</tr>
<tr>
<td>xdr_u_char()</td>
<td>Translate Unsigned Characters to/from XDR</td>
</tr>
<tr>
<td>xdr_u_int()</td>
<td>Translate Unsigned Integers to/from XDR</td>
</tr>
<tr>
<td>xdr_u_long()</td>
<td>Translate Unsigned Long Integers to/from XDR</td>
</tr>
<tr>
<td>xdr_u_short()</td>
<td>Translate Unsigned Short Integers to/from XDR</td>
</tr>
<tr>
<td>xdr_union()</td>
<td>Translate Discriminated Union to/from XDR</td>
</tr>
<tr>
<td>xdr_vector()</td>
<td>Translate Fixed-Length Arrays to/from XDR</td>
</tr>
<tr>
<td>xdr_void()</td>
<td>Translate Nothing to/from XDR</td>
</tr>
<tr>
<td>xdr_wrapstring()</td>
<td>Package RPC Message</td>
</tr>
<tr>
<td>xdrmem_create()</td>
<td>Initialize XDR Stream Object</td>
</tr>
</tbody>
</table>
### Table 2-17. XDR C Library Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdrrec_create()</td>
<td>Initialize XDR Stream</td>
</tr>
<tr>
<td>xdrrec_endofrecord()</td>
<td>Mark End of Record in XDR Stream</td>
</tr>
<tr>
<td>xdrrec_eof()</td>
<td>Check EOF on XDR Stream</td>
</tr>
<tr>
<td>xdrrec_skiprecord()</td>
<td>Skip Current Record in XDR Stream</td>
</tr>
<tr>
<td>xp prt_register()</td>
<td>Register Transport Handle</td>
</tr>
<tr>
<td>xp r t_unregister()</td>
<td>Unregister Service Transport Handle</td>
</tr>
<tr>
<td>xdrtdio_create()</td>
<td>Initialize XDR Stream Object</td>
</tr>
</tbody>
</table>
xdr_accepted_reply()
Encode RPC Reply Messages

Syntax
#include <RPC/rpc.h>
bool_t xdr_accepted_reply(
    XDR                       *xdrs,
    struct accepted_reply     *ar)

Description
xdr_accepted_reply() encodes the status of the RPC call. It is used to generate
RPC-style messages without using the RPC package.

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               User
Threads:             Safe

Parameters
xdrs  
is the service transport handle.

ar   
is the message accepted reply.
Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_array(
    XDR           *xdrs,
    caddr_t       *addrp,
    u_int         *sizep,
    u_int         maxsize,
    u_int         elsize,
    xdrproc_t     elproc)
```

Description

The `xdr_array()` filter primitive translates between variable-length arrays and their corresponding external representations. `elproc` translates between the array elements’ C form and their external representation.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `xdrs` is the stream.
- `addrp` is the address of the pointer to the array.
- `sizep` is the address of the element count of the array.
- `maxsize` specifies the maximum size of `sizep`.
- `elsize` is the size of each of the array’s elements.
- `elproc` is the filter.
Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_authunix_parms(
    XDR                       *xdrs,
    struct authunix_parms     *p)
```

Description

`xdr_authunix_parms()` externally describes credentials. This routine generates these credentials without using the authentication package.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xdrs` is the service transport handle.
- `p` is the authentication parameter.
xdr_bool()

Translate Booleans to/from XDR

Syntax
#include <RPC/rpc.h>
boots xdr_bool(
        XDR *xdrs,
        bool_t *bp)

Description
The xdr_bool() filter primitive translates between booleans (C integers) and their external representations. When encoding data, this filter produces values of either 1 or 0.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xdrs
    is the stream.
bp
    is the target boolean variable or constant.
Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_bytes(
    XDR *xdrs,
    char *cpp,
    u_int *sizep,
    u_int maxsize)
```

Description

The `xdr_bytes()` filter primitive translates between counted byte strings and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `xdrs` is the stream.
- `cpp` is the address of the string pointer.
- `sizep` is the length of the string.
- `maxsize` specifies the maximum size of `sizep`. 
Syntax
#include <RPC/rpc.h>
bool_t xdr_callhdr(
    XDR *xdrs,
    struct rpc_msg *cmsg)

Description
xdr_callhdr() encodes the static part of the call message header. It is used to generate RPC-style messages without using the RPC package.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xdrs
    is the service transport handle.

cmsg
    is the header portion of RPC message.
**Syntax**

```c
#include <RPC/rpc.h>
bool_t xdr_callmsg(
    XDR        *xdrs,
    struct rpc_msg *cmsg)
```

**Description**

`xdr_callmsg()` encodes an RPC call message. It is used to generate RPC-style messages without using the RPC package.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `xdrs` is the service transport handle.
- `cmsg` is the RPC call message.
xdr_char()

Translate Characters to/from XDR

Syntax
#include <RPC/rpc.h>
bool_t xdr_char(
    XDR     *xdrs,
    char    *cp)

Description
The xdr_char() filter primitive translates between C characters and their external representations.
If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes
Operating System:       OS-9 and OS-9 for 68K
State:                  User
Threads:                Safe

Parameters
xdrs
    is the stream.

cp
    is the target character.
    Encoded characters are not packed. They occupy four bytes each.
Syntax
#include <RPC/rpc.h>
void xdr_destroy(XDR *xdrs)

Description
The xdr_destroy() macro invokes the destroy routine associated with xdrs. Destruction usually involves freeing private data structures associated with the stream. Using xdrs after invoking xdr_destroy() is undefined.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xdrs
    is the stream.
**xdr_double()**

Translate Double Precision Numbers to/from XDR

**Syntax**
```
#include <RPC/rpc.h>
bool_t xdr_double(
    XDR       *xdrs,
    double    *dp)
```

**Description**
The `xdr_double()` filter primitive translates between C double precision numbers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**
- `xdrs`
  - is the stream.
- `dp`
  - is the target double precision variable.
xdr_enum()

Translate Enumerated Types to/from XDR

Syntax
#include <RPC/rpc.h>
bool_t xdr_enum(
    XDR        *xdrs,
    enum_t     *ep)

Description
The xdr_enum() filter primitive translates between C enumerations (actually integers) and their external representations.
If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xdrs
    is the stream.
ep
    is the target enumeration variable.
Syntax
#include <RPC/rpc.h>
bool_t xdr_float(
    XDR       *xdrs,
    float     *fp)

Description
The xdr_float() filter primitive translates between C floating point numbers and their external representations.
If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xdrs
    is the stream.

fp
    is the target float variable.
Syntax
#include <RPC/rpc.h>
void xdr_free(
    xdrproc_t proc,
    char *objp)

Description
xdr_free() is a generic freeing routine.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
proc
is the routine for the object being freed.

objp
is the pointer to the object itself.
The pointer passed to this routine is not freed, but what it points to is recursively freed.
Syntax

#include <RPC/rpc.h>

u_int xdr_getpos(XDR *xdrs)

Description

The `xdr_getpos()` macro invokes the get-position routine associated with `xdrs`. The routine returns an unsigned integer indicating the position of the byte stream. A desirable feature of streams is that simple arithmetic works with this number, although the stream instances need not guarantee this.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

xdrs

is the stream.
### Syntax

```c
#include <RPC/rpc.h>
long * xdr_inline(
    XDR     *xdrs,
    int     len)
```

### Description

The `xdr_inline` macro invokes the inline routine associated with `xdrs`. The routine returns a pointer to a contiguous piece of the stream’s buffer.

`xdr_inline()` may return 0 if it cannot allocate a contiguous piece of a buffer. Therefore, the behavior may vary among stream instances. It exists for efficiency.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- `xdrs` is the stream.
- `len` is the byte length of the desired buffer.
**Syntax**

```c
#include <RPC/rpc.h>
bool_t xdr_int(
    XDR      *xdrs,
    int      *ip)
```

**Description**

The `xdr_int()` filter primitive translates between C integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `xdrs` is the stream.
- `ip` is the target integer variable.
### Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_long(
    XDR      *xdrs,
    long     *lp)
```

### Description

The `xdr_long()` filter primitive translates between C long integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

### Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

### Parameters

- `xdrs` is the stream.
- `lp` is the target long variable.
Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_opaque(
    XDR *xdrs,
    caddr_t cp,
    u_int cnt)
```

Description

The `xdr_opaque()` filter primitive translates between fixed size opaque data and its external representation.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xdrs` is the stream.
- `cp` is the address of opaque object.
- `cnt` is the size of the opaque object in bytes.
xdr_opaque_auth()
Describe RPC Authentication Information

Syntax
#include <RPC/rpc.h>
bool_t xdr_opaque_auth(
    XDR                    *xdrs,
    struct opaque_auth     *ap)

Description
xdr_opaque_auth() describes RPC authentication information messages. It is used
to generate RPC-style messages without using the RPC package.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User
Threads:            Safe

Parameters
xdrs
    is the service transport handle.
ap
    is the opaque authentication structure.
**Syntax**

```c
#include <RPC/pmap_prot.h>
bool_t xdr_pmap(
    XDRS            *xdrs,
    struct pmap     *regs)
```

**Description**

`xdr_pmap()` externally describes parameters to various procedures. This routine generates these parameters without using the `pmap` interface.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `xdrs` is the service transport handle.
- `regs` is the portmap parameter.
xdr_pmaplist()

Describe Procedure Parameters and Port Maps

Syntax

```c
#include <RPC/pmap_prot.h>
bool_t xdr_pmaplist(
    XDR       *xdrs,
    struct pmaplist **rp)
```

Description

xdr_pmaplist() externally describes a list of port mappings. This routine generates these parameters without using the pmap interface.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `xdrs` is the service transport handle.
- `rp` is the portmap list.
xdr_pointer

Translate Pointer to/from XDR

Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_pointer(  
    XDR *xdrs,  
    char *objpp,  
    u_int obj_size,  
    xdrproc_t xdr_obj)
```

Description

xdr_pointer() serializes pointers. This routine can handle recursive data structures, such as binary trees or linked lists.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

xdrs
is the stream.

objpp
is a pointer to an object.

obj_size
is the size of object.

xdr_obj
is the XDR procedure to process target object.
xdr_reference()
Translate Pointers to/from XDR

Syntax
#include <RPC/rpc.h>

bool_t xdr_reference(
    XDR           *xdrs,
    caddr_t       *pp,
    u_int         size,
    xdrproc_t     proc)

Description
The xdr_reference() primitive provides pointer chasing within structures. proc filters the structure between its C form and its external representation.
If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:             User
Threads:           Safe

Parameters

xdrs
    is the stream.

pp
    is the address of the pointer.

size
    is the size of structure pointed to by pp.

proc
    is the filter.
**xdr_rejected_reply()**

*Encode Rejected RPC Message*

**Syntax**

```c
#include <RPC/rpc.h>
bool_t xdr_rejected_reply(
    XDR                       *xdrs,
    struct rejected_reply     *rr)
```

**Description**

`xdr_rejected_reply()` encodes the rejecting RPC message. It is used to generate RPC-style messages without using the RPC package.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `xdrs` is the service transport handle.
- `rr` is the message rejected reply.
**xdr_replymsg()**

Encode RPC Reply Message

**Syntax**

```c
#include <RPC/rpc.h>
bool_t xdr_replymsg(
    XDR *xdrs,
    struct rpc_msg *rmsg)
```

**Description**

`xdr_replymsg` encodes an RPC reply message. It is used to generate RPC-style messages without using the RPC package.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**

- `xdrs` is the service transport handle.
- `rmsg` is the RPC message.
#include <RPC/rpc.h>

```c
bool_t xdr_setpos(
    XDR       *xdrs,
    u_int     pos)
```

## Description

The `xdr_setpos()` macro invokes the set position routine associated with `xdrs`. This routine returns a value of 1 if the stream could be repositioned. Otherwise, it returns a value of 0.

## Attributes

**Operating System:** OS-9 and OS-9 for 68K

**State:** User

**Threads:** Safe

## Parameters

- `xdrs` is the stream.
- `pos` is the position value.

It is difficult to reposition some stream types. This routine may fail with one type of stream and succeed with another.
Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_short(
    XDR       *xdrs,
    short     *sp)
```

Description

The `xdr_short()` filter primitive translates between C short integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xdrs` is the stream.
- `sp` is the target short variable.
### Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_string(
    XDR       *xdrs,
    char      **cpp,
    u_int     maxsize)
```

### Description

The `xdr_string()` filter primitive translates between C strings and their corresponding external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

### Parameters

- **xdrs**
  - is the stream.
- **cpp**
  - is the address of the pointer to the string.
- **maxsize**
  - is the maximum size of the string.
Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_u_char(
    XDR        *xdrs,
    u_char     *cp)
```

Description

The `xdr_u_char()` filter primitive translates between C unsigned characters and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe

Parameters

- `xdrs`  
  is the stream.

- `cp`  
  is the target unsigned `char` variable.
Translate Unsigned Integers to/from XDR

**Syntax**
```
#include <RPC/rpc.h>
bool_t xdr_u_int(
    XDR       *xdrs,
    u_int     *up)
```

**Description**
The `xdr_u_int()` filter primitive translates between C integers and their external representations.
If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

**Attributes**
- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe

**Parameters**
- `xdrs`
  - is the stream.
- `up`
  - is the target unsigned integer variable.
xdr_u_long()

Translate Unsigned Long Integers to/from XDR

Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_u_long(
    XDR        *xdrs,
    u_long     *ulp)
```

Description

The `xdr_u_long()` filter primitive translates between C unsigned long integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:  OS-9 and OS-9 for 68K
State:  User
Threads:  Safe

Parameters

- **xdrs**
  - is the stream.

- **ulp**
  - is the target unsigned long variable.
xdr_u_short()

Translate Unsigned Short Integers to/from XDR

Syntax

```c
#include <RPC/rpc.h>
bool_t xdr_u_short(
    XDR         *xdrs,
    u_short     *usp)
```

Description

The `xdr_u_short()` filter primitive translates between C unsigned short integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `xdrs`
  - is the stream.
- `usp`
  - is the target unsigned short variable.
xdr_union()

Translate Discriminated Union to/from XDR

Syntax
#include <RPC/rpc.h>

bool_t xdr_union(
    XDR *xdrs,
    enum_t *dscmp,
    char *unp,
    struct xdr_discrim *choices,
    xdrproc_t dfault)

Description
The xdr_union() filter primitive translates between a discriminated C union and its corresponding external representation. It translates the discriminant of the union located at dscmp. This discriminant is always an integer. Then, the union located at unp is translated. Each structure contains an ordered pair of values. If the union’s discriminant is equal to the associated value, the routine translates the union. The end of the structure array is denoted by a routine of value 0. If the discriminant is not found in the array, the dfault procedure is called.

If successful, this routine returns a value of 1. Otherwise, it returns zero.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xdrs
    is the stream.

dscmp
    is the location of a union.

unp
    is the location of a union.

choices
    is the pointer to an array of structures.

default
    is the function to call if discriminant is not found (may be NULL).
xdr_vector()

Translate Fixed-Length Arrays to/from XDR

Syntax

```
#include <RPC/rpc.h>

bool_t xdr_vector(  
    XDR           *xdrs,  
    char          *basep,  
    u_int         nelem,  
    u_int         elemsize,  
    xdrproc_t     xdr_elem)
```

Description

The `xdr_vector()` filter primitive translates between fixed-length arrays and their corresponding external representations. `xdr_elem` translates between the array elements’ C form and their external representation.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

 Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- **xdrs**
  - is the stream.
- **basep**
  - is the address of the pointer to the array.
- **nelem**
  - is the element count of array.
- **elemsize**
  - is the size of each of the array’s elements.
- **xdr_elem**
  - is a filter.
xdr_void()

Translate Nothing to/from XDR

Syntax
#include <RPC/rpc.h>
bool_t xdr_void(void)

Description
xdr_void() always returns 1. It may be passed to routines that require a function parameter, where nothing is to be done.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
xdr_wrapstring()

Package RPC Message

Syntax

```
#include <RPC/rpc.h>

bool_t xdr_wrapstring(
    XDR      *xdrs,
    char     **cpp)
```

Description

The `xdr_wrapstring()` primitive calls `xdr_string()`. It is useful because the package passes a maximum of two routines as parameters, and one of the most frequently used primitives, `xdr_string()` requires three.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

`xdrs`

is the stream.

`cpp`

is the address of pointer to the string to convert.


**xdrmem_create()**

*Initialize XDR Stream Object*

**Syntax**

```c
#include <RPC/rpc.h>

void xdrmem_create(
    XDR             *xdrs,
    caddr_t         addr,
    u_int           size,
    enum xdr_op     op)
```

**Description**

`xdrmem_create()` initializes the stream object pointed to by `xdrs`. The stream’s data is written to memory or read from memory at location `addr`. `size` specifies the stream size.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

**Parameters**

- `xdrs` is the stream.
- `addr` is the address of a memory location.
- `size` is the maximum length of memory location.
- `op` determines the direction of the stream.
xdrrec_create()

Initialize XDR Stream

Syntax

```c
#include <RPC/rpc.h>
void xdrrec_create(
    XDR         *xdrs,
    u_int       sendsize,
    u_int       recvsize,
    caddr_t     handle,
    int         (*readit)(),
    int         (*writeit)()
)
```

Description

`xdrrec_create()` initializes the stream object pointed to by `xdrs`. Specifying values of 0 for `sendsize` or `recvsize` causes the system to choose suitable defaults. When a stream’s output buffer is full, `writeit` is called. Similarly, when a stream’s input buffer is empty, `readit` is called.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

- `xdrs` is the stream.
- `sendsize` is the size of stream’s outgoing data buffer.
- `recvsize` is the size of stream’s incoming data buffer.
- `handle` is the client handle.
- `readit` is the procedure for input empty condition.
- `writeit` is the procedure for output full condition.

This stream implements an intermediate record stream. Therefore, additional bytes in the stream provide record boundary information.
Syntax
#include <RPC/rpc.h>
bool_t xdrrec_endofrecord(
    XDR *xdrs,
    bool_t sendnow)

Description
xdrrec_endofrecord() marks the end of record in an XDR stream. This routine can be invoked only on record streams. The data in the output buffer is marked as a completed record. The output buffer is optionally written out if sendnow equals TRUE.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters
xdrs
    is the stream.

sendnow
    sends flag.
xdrrec_eof()

Check EOF on XDR Stream

Syntax

```c
#include <RPC/rpc.h>
bool_t xdrrec_eof(XDR *xdrs)
```

Description

`xdrrec_eof()` checks for an end-of-file condition on an XDR stream. After using the rest of the current record in the stream, this routine returns 1 if the stream has no more input. Otherwise, it returns a value of 0.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe

Parameters

- `xdrs`
  - is a pointer to a stream.
xdrrec_skiprecord()

Skip Current Record in XDR Stream

Syntax

#include <RPC/rpc.h>
bool_t xdrrec_skiprecord(XDR *xdrs)

Description

xdrrec_skiprecord() skips the rest of the current record in the stream's input buffer. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

xdrs

is a pointer to a stream.
Syntax

#include <RPC/rpc.h>

void xprt_register(SVCXPRT *xprt)

Description

xprt_register() registers the transport handle xprt. After service transport handles are created, they should register themselves with the service package. Service implementors usually do not need to use this routine.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

Parameters

xprt is a service transport handle.
syntax
#include <RPC/rpc.h>
void xprt_unregister(SVCXPRT *xprt)

description
xprt_unregister() unregisters a service transport handle. Before a service transport handle is destroyed, it should unregister itself with the service package. Service implementors usually do not need to use this routine.

attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe

parameters
xprt
    is a service transport handle.
Syntax

```
#include <RPC/rpc.h>
void xdrstdio_create(
    XDR             *xdrs,
    FILE            *file,
    enum xdr_op     op)
```

Description

`xdrstdio_create()` initializes the stream object pointed to by `xdrs`. The stream data is written to, or read from, `file`.

Attributes

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User
- **Threads**: Safe

Parameters

- **xdrs**: is a stream.
- **file**: is the name of file containing stream data.
- **op**: determines the direction of the stream.
endprotoent 56
endresolvent 57
endroutent 58
endservent 59

entry
  Add Interface 104
  add network to inetdb 105
  network, get 75
  protocol, get 76
Entry, Add Interface 104
Entry, Add Route 109
Entry, Delete Route 50

Errors
  EOS_MNF 126

F
  file, hosts 64

G
  get
    network entry 75
    protocol entry 76
get_mymaddress 305
getaddrinfo 60
gethostbyaddr 62
gethostbyname 64
gethostbyname() 64
gethostbyname2 65
gethostname() 126
gethostent 67
gethostname 126
gethostname() 126
getifaddrs 68
getinetdent 69
getintent 70
getnameinfo 71
getnetbyaddr 73
getnetbyname 74
getnetent 75
getpeername 127
getprotobynumber 77
getprotoent 78
getresolvent 79

Gets Name of Current Host 126
getcurrenthostname 126
Host, Gets Name of Current 126
hostconfent 14
hostent 13, 64
hosts file 64
htonl 83
htons 84

I
  if_indextoname 85
  if_nameindex 86
    return structure array 86
if_nametoindex 87
inet_addr 88
inet_aton 89
inet_lnaof 90
inet_makeaddr 91
inet_netof 92
inet_network 93
inet_ntoa 94
inet_ntop 95
inet_pton 97
inetdb, linking to 42

Interface Entry, Add 104

Internet Data Module Structures
  hostconfent 14
  hostent 13
  n_ifaliasreq 15
  n_ifaliasreq6 16
  n_ifnet 17
  netent 19
  protoent 20
  resolvent 21
  rtreq 22, 23
  servent 24
ip_start 132
ite_ctl_addrset 198
ite_ctl_answer 200
ite_ctl_connect 202
ite_ctl_connstat 204
ite_ctl_disconnect 205
ite_ctl_rcvrasgn 206
ite_ctl_rcvrrmv 208
ite_data_avail_asgn 209
ite_data_avail_rmv 211
ite_data_read 212
ite_data_readmbuf 214
ite_data_ready 216
ite_data_recvfrom 217
ite_data_sendto 219
ite_data_write 221
ite_data_writembuf 223
ite_dev_attach 224
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ite_dev_detach</td>
<td>225</td>
</tr>
<tr>
<td>ite_dev_getmode</td>
<td>226</td>
</tr>
<tr>
<td>ite_dev_gettype</td>
<td>228</td>
</tr>
<tr>
<td>ite_dev_setmode</td>
<td>230</td>
</tr>
<tr>
<td>ite_fehangup_asgn</td>
<td>231</td>
</tr>
<tr>
<td>ite_fehangup_rmv</td>
<td>232</td>
</tr>
<tr>
<td>ite_linkdown_asgn</td>
<td>233</td>
</tr>
<tr>
<td>ite_linkdown_rmv</td>
<td>234</td>
</tr>
<tr>
<td>ite_linkup_asgn</td>
<td>235</td>
</tr>
<tr>
<td>ite_linkup_rmv</td>
<td>236</td>
</tr>
<tr>
<td>ite_path_clone</td>
<td>237</td>
</tr>
<tr>
<td>ite_path_close</td>
<td>239</td>
</tr>
<tr>
<td>ite_path_dup</td>
<td>240</td>
</tr>
<tr>
<td>ite_path_open</td>
<td>241</td>
</tr>
<tr>
<td>ite_path_pop</td>
<td>243</td>
</tr>
<tr>
<td>ite_path_profileget</td>
<td>244</td>
</tr>
<tr>
<td>ite_path_profileset</td>
<td>246</td>
</tr>
<tr>
<td>ite_path_push</td>
<td>248</td>
</tr>
</tbody>
</table>

**ITEM library functions**

- ite_ctl_addrset 198
- ite_ctl_answer 200
- ite_ctl_connect 202
- ite_ctl_connstat 204
- ite_ctl_disconnect 205
- ite_ctl_rcvrasgn 206
- ite_ctl_rcvrrmv 208
- ite_data_avail_asgn 209
- ite_data_avail_rmv 211
- ite_data_read 212
- ite_data_readmbuf 214
- ite_data_ready 216
- ite_data_recvfrom 217
- ite_data_sendto 219
- ite_data_write 221
- ite_data_writembuf 223
- ite_dev_attach 224
- ite_dev_detach 225
- ite_dev_getmode 226
- ite_dev_gettype 228
- ite_dev_setmode 230
- ite_fehangup_asgn 231
- ite_fehangup_rmv 232
- ite_linkdown_asgn 233
- ite_linkdown_rmv 234
- ite_linkup_asgn 235
- ite_linkup_rmv 236
- ite_path_clone 237
- ite_path_close 239
- ite_path_dup 240
- ite_path_open 241
- ite_path_pop 243
- ite_path_profileget 244
- ite_path_profileset 246

**ndblib.l**

- ndb_create_ndbmod 118

**netdb.l**

- delhostbyname 45
- delhostconfent 46
- delintbyname 47
- delresolvent 48
- delroutent 49
- delroutent6 50
- delservbyname 51
- endhostconfent 52
- endhostent 53
- endintent 54
endnetent 55
endprotoent 56
endresolvent 57
endroutent 58
endservent 59
getaddrinfo 60
gethostbyaddr 62
gethostbyname 64
gethostbyname2 65
gethostconfent 66
gethostent 67
getifaddrs 68
getinetd 69
getintent 70
getnameinfo 71
getnetbyaddr 73
getnetbyname 74
getnetent 75
getprotobynumber 77
getprotonent 78
getresolvent 79
getservbyport 80
getservbyname 80
getservent 82
htnl 83
htons 84
if_indextoname 85
if_nameindex 86
if_nametoindex 87
inet_addr 88
inet_aton 89
inet_lnaof 90
inet_makeaddrt 91
inet_netof 92
inet_network 93
inet_ntoa 94
inet_ntop 95
inet_pton 97
ntohl 99
ntohs 100
puthostconfent 101
puthostent 102
putintent 103
putintent6 104
putnetent 105
putprotonent 106
putresolvent 107
putroutent 108
putroutent6 109
res_cancel 111
sethostent 112
setnetent 113
setprotonent 114
setservent 115
listen 133

M
m_adj 259
m_cat 260
m_copy 261
m_deq 262
m_enq 263
m_flush 264
m_free 265
m_free_m 266
m_free_p 267
m_free_q 268
m_get 269
m_getn 270
m_len_p 271
m_move 272
m_msize 273
m_pad 274
M_PREPEND 276
m_ptod 275
m_pullup 277
m_unpad 279
mbuf 38
mbuf Data Structure
  mbuf 38
mbuf functions
  m_adj 259
  m_cat 260
  m_copy 261
  m_deq 262
  m_enq 263
  m_move 272
  m_msize 273
  m_pad 274
  M_PREPEND 276
  m_ptod 275
  m_pullup 277
  m_unpad 279
mtod 278
mtod 278

<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>n_ifaliasreq</td>
</tr>
<tr>
<td>n_ifaliasreq6</td>
</tr>
<tr>
<td>n_INET</td>
</tr>
<tr>
<td>Name of Current Host, Gets</td>
</tr>
<tr>
<td>names, get host</td>
</tr>
<tr>
<td>ndb_create_ndbmod</td>
</tr>
<tr>
<td>ndblib.l library functions</td>
</tr>
<tr>
<td>ndb_create_ndbmod</td>
</tr>
<tr>
<td>netdb.l library functions</td>
</tr>
<tr>
<td>delhostbyname</td>
</tr>
<tr>
<td>delhostconfent</td>
</tr>
<tr>
<td>delintbyname</td>
</tr>
<tr>
<td>delresolver</td>
</tr>
<tr>
<td>delroutent</td>
</tr>
<tr>
<td>delroutent6</td>
</tr>
<tr>
<td>delservbyname</td>
</tr>
<tr>
<td>endhostconfent</td>
</tr>
<tr>
<td>endhostent</td>
</tr>
<tr>
<td>endintent</td>
</tr>
<tr>
<td>endnetent</td>
</tr>
<tr>
<td>endprotoent</td>
</tr>
<tr>
<td>endresolver</td>
</tr>
<tr>
<td>endroutent</td>
</tr>
<tr>
<td>endservent</td>
</tr>
<tr>
<td>gethostbyname</td>
</tr>
<tr>
<td>gethostbyaddr</td>
</tr>
<tr>
<td>gethostbyname2</td>
</tr>
<tr>
<td>gethostconfent</td>
</tr>
<tr>
<td>gethostent</td>
</tr>
<tr>
<td>gethostbyaddr</td>
</tr>
<tr>
<td>getifaddrs</td>
</tr>
<tr>
<td>getinetdent</td>
</tr>
<tr>
<td>getintent</td>
</tr>
<tr>
<td>getnameinfo</td>
</tr>
<tr>
<td>getnetbyaddr</td>
</tr>
<tr>
<td>getnetbyname</td>
</tr>
<tr>
<td>getnetent</td>
</tr>
<tr>
<td>getprotobynumber</td>
</tr>
<tr>
<td>getprotoent</td>
</tr>
<tr>
<td>getresolver</td>
</tr>
<tr>
<td>getservbyname</td>
</tr>
<tr>
<td>getservbyport</td>
</tr>
<tr>
<td>getservent</td>
</tr>
<tr>
<td>htonl</td>
</tr>
<tr>
<td>htons</td>
</tr>
<tr>
<td>if_indextoname</td>
</tr>
<tr>
<td>if_nametoindex</td>
</tr>
<tr>
<td>if_nameindex</td>
</tr>
<tr>
<td>inet_addr</td>
</tr>
<tr>
<td>inet_pton</td>
</tr>
<tr>
<td>inet_lnaof</td>
</tr>
<tr>
<td>inet_makeaddr</td>
</tr>
<tr>
<td>inet_netof</td>
</tr>
<tr>
<td>inet_network</td>
</tr>
<tr>
<td>inet_ntoa</td>
</tr>
<tr>
<td>inet_ntop</td>
</tr>
<tr>
<td>ntohl</td>
</tr>
<tr>
<td>ntohs</td>
</tr>
<tr>
<td>puthostconfent</td>
</tr>
<tr>
<td>puthostent</td>
</tr>
<tr>
<td>putintent</td>
</tr>
<tr>
<td>putintent6</td>
</tr>
<tr>
<td>putnetent</td>
</tr>
<tr>
<td>putprotoent</td>
</tr>
<tr>
<td>putresolver</td>
</tr>
<tr>
<td>putroutent</td>
</tr>
<tr>
<td>putroutent6</td>
</tr>
<tr>
<td>res_cancel</td>
</tr>
<tr>
<td>sethostent</td>
</tr>
<tr>
<td>setnetent</td>
</tr>
<tr>
<td>setprotoent</td>
</tr>
<tr>
<td>setservent</td>
</tr>
<tr>
<td>netdb_resolv</td>
</tr>
<tr>
<td>netent</td>
</tr>
<tr>
<td>network</td>
</tr>
<tr>
<td>entry, add to inetdb</td>
</tr>
<tr>
<td>entry, get</td>
</tr>
<tr>
<td>host entry</td>
</tr>
<tr>
<td>networking files</td>
</tr>
<tr>
<td>hosts</td>
</tr>
<tr>
<td>ntohl</td>
</tr>
<tr>
<td>ntohs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS and Conv functions</td>
</tr>
<tr>
<td>_os_getstat</td>
</tr>
<tr>
<td>_os_setstat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>per-path static storage functions</td>
</tr>
<tr>
<td>pps_add_entry</td>
</tr>
<tr>
<td>pps_chg_updrvr</td>
</tr>
<tr>
<td>pps_del_entry</td>
</tr>
<tr>
<td>pps_find_entry</td>
</tr>
<tr>
<td>pps_find_entry_by_offset</td>
</tr>
<tr>
<td>pmap_getmaps</td>
</tr>
<tr>
<td>pmap_getport</td>
</tr>
<tr>
<td>pmap_rmtcall</td>
</tr>
<tr>
<td>pmap_set</td>
</tr>
<tr>
<td>pmap_unset</td>
</tr>
<tr>
<td>Point-to-Point protocol functions</td>
</tr>
<tr>
<td>ppp_auth_add_chap</td>
</tr>
<tr>
<td>ppp_auth_add_pap</td>
</tr>
</tbody>
</table>
Point-to-Point Protocol Structures
auth_handle 26
ppp_conninfo 27
ppp_error 28
ppp_get_asynca_params 177
ppp_get_options 178
ppp_get_params 179
ppp_get_statistics 180
ppp_hdlc_stats 29
ppp_init 182
ppp_modem_p 30
ppp_open 183
ppp_param_block 33
ppp_reset_statistics 184
ppp_set_asynca_params 185
ppp_set_options 187
ppp_start 188
ppp_term 189
pps_add_entry 191
pps_chg_updrvr 192
pps_del_entry 194
pps_find_entry 193
pps_find_entry_by_offset 195
R
recv 134
recvfrom 136
recvmsg 138
registerrpc 312
res_cancel 111
resolvconent 101
puthostconfent 101
puthostent 102
putintent 103
putintent() 104
putintent6 104
putnetent 105
putprotoent 106
putresolvent 107
putroutent 108
putroutent() 109
putroutent6 109
return
array of if_nameindex structures 86
Route Entry, Add 109
Route Entry, Delete 50
RPC functions
auth_destroy 282
authnone_create 283
authunix_create 284
authunix_create_default 285
clnrpc 286
clnt_broadcast 288
clnt_call 290
clnt_control 291
clnt_create 292
clnt_destroy 293
clnt_freeres 294
clnt_geterr 295
clnt_pcreateerror 296
clnt_perrno 297
clnt_perror 298
clnt_spcreateerror 299
clnt_sperrno 300
clnt_verror 301
clntraw_create 302
clnttcp_create 303
clntudp_create 304
get_myaddress 305
pmap_getmaps 306
pmap_getport 307
pmap_rmtcall 308
pmap_set 310
pmap_unset 311
registerrpc 312
svc_destroy 313
svc_freeargs 314
svc_getargs 315
svc_getcaller 316
svc_getreq 317
svc_getreqset 318
svc_register 319
svc_run 320
svc_sendreply 321
svc_unregister 322
svcerr_auth 323
svcerr_decode 324
svcerr_noproc 325
svcerr_noprog 326
svcerr_program 327
svcerr_systemerr 328
svcerr_weakauth 329
svcfd_create 330
svcraw_create 331
svctcp_create 332
svctudp_create 333
RPC/XDR Programming Structures
auth.h 35
clnt.h 36
svc.h 36
xdr.h 37
rteq 22, 23
S
send 139
sendmsg 143
sendto 141
servent 24
sethostent 112
sethostname 144
setnetent 113
setprotoent 114
setservent 115
setsockopt 145
shutdown 148
sockaddr 116
socket 149
socket functions, Berkeley
accept 121
bind 123
connect 124
gethostbyname 126
getpeername 127
getsockname 128
getsockopt 129
ip_start 132
listen 133
recv 134
recvfrom 136
recvmsg 138
send 139
sendmsg 143
sendto 141
sethostname 144
setsockopt 145
socket 149
socket functions, Berkeley
shutdown 148
static storage functions
pps_add_entry 191
pps_chg_updrvr 192
pps_del_entry 194
pps_find_entry 193
pps_find_entry_by_offset 195
structures
Internet Data Module
hostconfent 14
hostent 13
n_ifaliasreq 15
n_ifaliasreq6 16
n_ifnet 17
netent 19
protoent 20
resolvent 21
rteq 22, 23
servent 24
OS-9 Network Programming Reference

mbuf Data
mbuf 38

PC/XDR Programming
auth.h 35
cln.h 36
csv.h 36
xdr.h 37

Point-to-Point Protocol
auth_handle 26
ppp_conninfo 27
ppp_error 28
ppp_hdlc_stats 29
ppp_modem_p 30
ppp_param_block 33
return array of if_nameindex 86

svc.h 36
svc_destroy 313
svc_freeargs 314
svc_getargs 315
svc_getcaller 316
svc_getreq 317
svc_getreqset 318
svc_register 319
svc_run 320
svc_sendreply 321
svc_unregister 322
svcerr_auth 323
svcerr_decode 324
svcerr_noproc 325
svcerr_noprog 326
svcerr_progrers 327
svcerr_systemerr 328
svcerr_weakauth 329
svcfd_create 330
svcraw_create 331
svctcp_create 332
svcudp_create 333

X

XDR functions
xdr_accepted_reply 337
xdr_array 338
xdr_authunix_parms 339
xdr_bool 340
xdr_bytes 341
xdr_callhdr 342
xdr_callmsg 343
xdr_char 344
xdr_destroy 345
xdr_double 346
xdr_enum 347
xdr_float 348
xdr_free 349
xdr_getpos 350
xdr_inline 351
xdr_int 352
xdr_long 353
xdr_opaque 354
xdr_opaque_auth 355
xdr_pmap 356
xdr_pmaplist 357
xdr_pointer 358
xdr_reference 359
xdrRejected_reply 360
xdr_replmsg 361
xdr_setpos 362
xdr_short 363
xdr_string 364
xdr_u_char 365
xdr_u_int 366
xdr_u_long 367
xdr_u_short 368
xdr_union 369
xdr_vector 370
xdr_void 371
xdr_wrapstring 372
xdrmem_create 373
xdrrec_create 374
xdrrec_endofrecord 375
xdrrec_eof 376
xdrrec_skiprecord 377
xdrstdio_create 378
xprtr_register 379
xprtr_unregister 379
xdr.h 37
xdr_accepted_reply 337
xdr_array 338
xdr_authunix_parms 339
xdr_bool 340
xdr_bytes 341
xdr_callhdr 342
xdr_callmsg 343
xdr_char 344
xdr_destroy 345
xdr_double 346
xdr_enum 347
xdr_float 348
xdr_free 349
xdr_getpos 350
xdr_inline 351
xdr_int 352
xdr_long 353
xdr_opaque 354
xdr_opaque_auth 355
xdr_pmap 356
xdr_pmaplist 357
xdr_pointer 358
xdr_reference 359
xdr_rejected_reply 360
xdr_replymsg 361
xdr_setpos 362
xdr_short 363
xdr_string 364
xdr_u_char 365
xdr_u_int 366
xdr_u_long 367
xdr_u_short 368
xdr_union 369
xdr_vector 370
xdr_void 371
xdr_wrapstring 372
xdrmem_create 373
xdrrec_create 374
xdrrec_endofrecord 375
xdrrec_eof 376
xdrrec_skiprecord 377
xdrstdio_create 380
xppt_register 378
xppt_unregister 379