DTS Application Library

0.2.3

Generated by Doxygen 1.8.5

Wed Oct 23 2013 10:44:30

Contents

1 Distrotech Application Library Manual **1**

1.1 Introduction . . . 1

1.2 Further information . . . 1

1.3 Copyright information. . . . 1

2 Application Startup **3**

2.1 Using helper macro instead of main() . . . 3

3 Referenced Lockable Objects **5**

3.1 Introduction . . . 5

3.2 Other referenced object functions. . . . 6

3.3 Internal workings. . . . 6

3.4 Referenced Lockable Objects With Classes (C++) . . . 7

3.5 Downsides . . . 7

4 Hashed Bucket Lists **9**

4.1 Introduction . . . 9

4.2 Usage of hashed bucket lists . . . 10

5 Thread Interface **11**

5.1 Introduction . . . 11

5.2 Creating A Thread . . . 11

6 Socket Interface **13**

6.1 Introduction . . . 13

6.2 SSL Support . . . 13

6.3 Socket Creation . . . 13

6.4 Starting A Socket . . . 13

6.5 Reading/Writeing To Sockets . . . 14

6.6 Unix Domain Sockets . . . 14

6.7 Multicast Sockets . . . 14

6.8 Example Code . . . 14

7 Socket Example (Echo Server/Client) **15**

7.1 Details . . . 15
7.2 Annotation . . . 15
7.3 Code . . . 15
7.4 Output . . . 17

8 Todo List **19**

9 Module Index **21**
9.1 Modules . . . 21

10 Data Structure Index **23**
10.1 Data Structures . . . 23

11 File Index **25**
11.1 File List . . . 25

12 Module Documentation **27**
12.1 Distrotech Application Library . . . 27
12.1.1 Detailed Description . . . 28
12.1.2 Macro Definition Documentation . . . 29
12.1.2.1 ALLOC_CONST . . . 29
12.1.2.2 FRAMEWORK_MAIN . . . 29
12.1.3 Typedef Documentation . . . 30
12.1.3.1 frameworkfunc . . . 30
12.1.3.2 syssighandler . . . 30
12.1.4 Enumeration Type Documentation . . . 30
12.1.4.1 framework_flags . . . 30
12.1.5 Function Documentation . . . 31
12.1.5.1 daemonize . . . 31
12.1.5.2 framework_init . . . 31
12.1.5.3 framework_mkcore . . . 32
12.1.5.4 lockpidfile . . . 33
12.1.5.5 printgnu . . . 34
12.2 Referenced Lockable Objects . . . 35
12.2.1 Detailed Description . . . 36
12.2.2 Macro Definition Documentation . . . 36
12.2.2.1 clearflag . . . 36
12.2.2.2 DTS_OJBREF_CLASS . . . 36
12.2.2.3 REFOBJ_MAGIC . . . 37
12.2.2.4 refobj_offset . . . 37
12.2.2.5 setflag . . . 37
12.2.2.6 testflag . . . 37

12.2.3 Typedef Documentation . . . 37
12.2.3.1 objdestroy . . . 37
12.2.4 Function Documentation . . . 37
12.2.4.1 objalloc . . . 37
12.2.4.2 objchar . . . 38
12.2.4.3 objcnt . . . 39
12.2.4.4 objlock . . . 40
12.2.4.5 objref . . . 41
12.2.4.6 objsize . . . 41
12.2.4.7 objtrylock . . . 42
12.2.4.8 objunlock . . . 42
12.2.4.9 objunref . . . 43
12.3 Hashed bucket linked lists of referenced objects . . . 45
12.3.1 Detailed Description . . . 46
12.3.2 Typedef Documentation . . . 46
12.3.2.1 blist_cb . . . 46
12.3.2.2 blisthash . . . 46
12.3.3 Function Documentation . . . 47
12.3.3.1 addtobucket . . . 47
12.3.3.2 bucket_list_cnt . . . 48
12.3.3.3 bucket_list_find_key . . . 49
12.3.3.4 bucketlist_callback . . . 49
12.3.3.5 create_bucketlist . . . 50
12.3.3.6 init_bucket_loop . . . 51
12.3.3.7 next_bucket_loop . . . 52
12.3.3.8 remove_bucket_item . . . 53
12.3.3.9 remove_bucket_loop . . . 54
12.4 Posix thread interface . . . 55
12.4.1 Detailed Description . . . 56
12.4.2 Typedef Documentation . . . 56
12.4.2.1 threadcleanup . . . 56
12.4.2.2 threadfunc . . . 56
12.4.2.3 threadsighandler . . . 56
12.4.3 Enumeration Type Documentation . . . 57
12.4.3.1 thread_option_flags . . . 57
12.4.3.2 threadopt . . . 57
12.4.4 Function Documentation . . . 58
12.4.4.1 framework_mkthread . . . 58
12.4.4.2 framework_threadok . . . 59
12.4.4.3 jointhreads . . . 59

12.4.4.4 startthreads . . . 60
12.4.4.5 stopthreads . . . 61
12.4.4.6 thread_signal . . . 61
12.4.5 Variable Documentation . . . 62
12.4.5.1 thread_can_start . . . 62
12.4.5.2 threads . . . 62
12.5 Network socket interface . . . 63
12.5.1 Detailed Description . . . 64
12.5.2 Typedef Documentation . . . 64
12.5.2.1 socketrecv . . . 64
12.5.3 Enumeration Type Documentation . . . 65
12.5.3.1 sock_flags . . . 65
12.5.4 Function Documentation . . . 65
12.5.4.1 accept_socket . . . 65
12.5.4.2 close_socket . . . 66
12.5.4.3 make_socket . . . 67
12.5.4.4 sockaddr2ip . . . 67
12.5.4.5 sockbind . . . 68
12.5.4.6 sockconnect . . . 69
12.5.4.7 socketclient . . . 69
12.5.4.8 socketread . . . 70
12.5.4.9 socketread_d . . . 70
12.5.4.10 socketserver . . . 72
12.5.4.11 socketwrite . . . 72
12.5.4.12 socketwrite_d . . . 73
12.5.4.13 tcpbind . . . 75
12.5.4.14 tcpconnect . . . 75
12.5.4.15 udpbind . . . 76
12.5.4.16 udpconnect . . . 76
12.6 SSL socket support . . . 77
12.6.1 Detailed Description . . . 78
12.6.2 Macro Definition Documentation . . . 78
12.6.2.1 COOKIE_SECRET_LENGTH . . . 78
12.6.3 Typedef Documentation . . . 78
12.6.3.1 ssldata . . . 78
12.6.4 Enumeration Type Documentation . . . 78
12.6.4.1 SSLFLAGS . . . 78
12.6.5 Function Documentation . . . 79
12.6.5.1 dtls_listenssl . . . 79
12.6.5.2 dtlshandltimeout . . . 80

12.6.5.3 dtlstimeout . . . 80
12.6.5.4 dtlsv1_init . . . 81
12.6.5.5 dtsl_serveropts . . . 81
12.6.5.6 ssl_shutdown . . . 82
12.6.5.7 sslstartup . . . 82
12.6.5.8 sslv2_init . . . 83
12.6.5.9 sslv3_init . . . 83
12.6.5.10 startsslclient . . . 83
12.6.5.11 tlsaccept . . . 84
12.6.5.12 tlsv1_init . . . 84
12.7 Unix domain sockets . . . 86
12.7.1 Detailed Description . . . 86
12.7.2 Function Documentation . . . 86
12.7.2.1 unixsocket_client . . . 86
12.7.2.2 unixsocket_server . . . 88
12.8 Multicast sockets . . . 90
12.8.1 Detailed Description . . . 90
12.8.2 Function Documentation . . . 90
12.8.2.1 mcast4_ip . . . 90
12.8.2.2 mcast6_ip . . . 90
12.8.2.3 mcast_socket . . . 91
12.9 Linux network interface functions . . . 94
12.9.1 Detailed Description . . . 95
12.9.2 Enumeration Type Documentation . . . 95
12.9.2.1 ipv4_score . . . 95
12.9.2.2 ipv6_score . . . 95
12.9.3 Function Documentation . . . 96
12.9.3.1 closenetlink . . . 96
12.9.3.2 create_kernmac . . . 96
12.9.3.3 create_kernvlan . . . 97
12.9.3.4 create_tun . . . 98
12.9.3.5 delete_kernmac . . . 98
12.9.3.6 delete_kernvlan . . . 99
12.9.3.7 get_iface_index . . . 99
12.9.3.8 get_ifipaddr . . . 100
12.9.3.9 ifdown . . . 101
12.9.3.10 ifhwaddr . . . 101
12.9.3.11 ifrename . . . 102
12.9.3.12 ifup . . . 102
12.9.3.13 interface_bind . . . 103

12.9.3.14 randhwaddr . . . 103
12.9.3.15 set_interface_addr . . . 104
12.9.3.16 set_interface_flags . . . 104
12.9.3.17 set_interface_ipaddr . . . 105
12.9.3.18 set_interface_name . . . 106
12.10 INI Style config file Interface . . . 108
12.10.1 Detailed Description . . . 109
12.10.2 Typedef Documentation . . . 109
12.10.2.1 config_catcb . . . 109
12.10.2.2 config_entrycb . . . 109
12.10.2.3 config_filecb . . . 109
12.10.3 Function Documentation . . . 109
12.10.3.1 config_cat_callback . . . 109
12.10.3.2 config_entry_callback . . . 110
12.10.3.3 config_file_callback . . . 110
12.10.3.4 get_category_loop . . . 110
12.10.3.5 get_category_next . . . 111
12.10.3.6 get_config_category . . . 111
12.10.3.7 get_config_entry . . . 112
12.10.3.8 get_config_file . . . 112
12.10.3.9 process_config . . . 113
12.10.3.10 unrefconfigfiles . . . 114
12.11 Radius client interface . . . 115
12.11.1 Detailed Description . . . 116
12.11.2 Macro Definition Documentation . . . 116
12.11.2.1 RAD_ATTR_ACCTID . . . 116
12.11.2.2 RAD_ATTR_EAP . . . 116
12.11.2.3 RAD_ATTR_MESSAGE . . . 117
12.11.2.4 RAD_ATTR_NAS_IP_ADDR . . . 117
12.11.2.5 RAD_ATTR_NAS_PORT . . . 117
12.11.2.6 RAD_ATTR_PORT_TYPE . . . 117
12.11.2.7 RAD_ATTR_SERVICE_TYPE . . . 117
12.11.2.8 RAD_ATTR_USER_NAME . . . 117
12.11.2.9 RAD_ATTR_USER_PASSWORD . . . 117
12.11.2.10 RAD_AUTH_HDR_LEN . . . 117
12.11.2.11 RAD_AUTH_PACKET_LEN . . . 117
12.11.2.12 RAD_AUTH_TOKEN_LEN . . . 118
12.11.2.13 RAD_MAX_PASS_LEN . . . 118
12.11.3 Typedef Documentation . . . 118
12.11.3.1 radius_cb . . . 118

12.11.3.2 radius_packet . . . 118
12.11.4 Enumeration Type Documentation . . . 118
12.11.4.1 RADIUS_CODE . . . 118
12.11.5 Function Documentation . . . 119
12.11.5.1 add_radserver . . . 119
12.11.5.2 addradattrint . . . 119
12.11.5.3 addradattrip . . . 119
12.11.5.4 addradattrstr . . . 120
12.11.5.5 new_radpacket . . . 120
12.11.5.6 radius_attr_first . . . 121
12.11.5.7 radius_attr_next . . . 121
12.11.5.8 send_radpacket . . . 121
12.12 Micelaneous utilities. . . . 123
12.12.1 Detailed Description . . . 123
12.12.2 Function Documentation . . . 124
12.12.2.1 b64enc . . . 124
12.12.2.2 b64enc_buf . . . 125
12.12.2.3 checksum . . . 125
12.12.2.4 checksum_add . . . 126
12.12.2.5 genrand . . . 126
12.12.2.6 ltrim . . . 127
12.12.2.7 rtrim . . . 128
12.12.2.8 seedrand . . . 128
12.12.2.9 strlenzero . . . 129
12.12.2.10 touch . . . 129
12.12.2.11 trim . . . 130
12.12.2.12 vtontp64 . . . 130
12.12.2.13 verifysum . . . 130
12.13 Hashing and digest functions . . . 132
12.13.1 Detailed Description . . . 132
12.14 MD5 Hashing and digest functions . . . 133
12.14.1 Detailed Description . . . 133
12.14.2 Function Documentation . . . 133
12.14.2.1 md5cmp . . . 133
12.14.2.2 md5hmac . . . 133
12.14.2.3 md5sum . . . 134
12.14.2.4 md5sum2 . . . 134
12.15 SHA1 Hashing and digest functions . . . 135
12.15.1 Detailed Description . . . 135
12.15.2 Function Documentation . . . 135

12.15.2.1 sha1cmp . . . 135
12.15.2.2 sha1hmac . . . 135
12.15.2.3 sha1sum . . . 136
12.15.2.4 sha1sum2 . . . 136
12.16 SHA2-256 Hashing and digest functions . . . 137
12.16.1 Detailed Description . . . 137
12.16.2 Function Documentation . . . 137
12.16.2.1 sha256cmp . . . 137
12.16.2.2 sha256hmac . . . 137
12.16.2.3 sha256sum . . . 138
12.16.2.4 sha256sum2 . . . 138
12.17 SHA2-512 Hashing and digest functions . . . 139
12.17.1 Detailed Description . . . 139
12.17.2 Function Documentation . . . 139
12.17.2.1 sha512cmp . . . 139
12.17.2.2 sha512hmac . . . 139
12.17.2.3 sha512sum . . . 140
12.17.2.4 sha512sum2 . . . 140
12.18 IPv4 and IPv6 functions . . . 141
12.18.1 Detailed Description . . . 141
12.18.2 Enumeration Type Documentation . . . 141
12.18.2.1 ipversion . . . 141
12.18.3 Function Documentation . . . 142
12.18.3.1 inet_lookup . . . 142
12.18.3.2 packetchecksum . . . 143
12.19 IPv4 functions . . . 145
12.19.1 Detailed Description . . . 145
12.19.2 Function Documentation . . . 146
12.19.2.1 check_ipv4 . . . 146
12.19.2.2 cidrcnt . . . 146
12.19.2.3 cidrtosn . . . 146
12.19.2.4 getbcaddr . . . 147
12.19.2.5 getfirstaddr . . . 148
12.19.2.6 getlastaddr . . . 148
12.19.2.7 getnetaddr . . . 149
12.19.2.8 ipv4checksum . . . 150
12.19.2.9 ipv4icmpchecksum . . . 151
12.19.2.10 ipv4tcpchecksum . . . 151
12.19.2.11 ipv4udpchecksum . . . 152
12.19.2.12 packetchecksumv4 . . . 153

12.19.2.13 reservedip . . . 153
12.19.2.14 score_ipv4 . . . 154
12.20 IPv6 functions . . . 156
12.20.1 Detailed Description . . . 156
12.20.2 Function Documentation . . . 156
12.20.2.1 checkipv6mask . . . 156
12.20.2.2 eui48to64 . . . 157
12.20.2.3 get_ip6_addrprefix . . . 157
12.20.2.4 ipv6to4prefix . . . 158
12.20.2.5 packetchecksumv6 . . . 158
12.20.2.6 score_ipv6 . . . 159
12.21 File utility functions . . . 160
12.21.1 Detailed Description . . . 160
12.21.2 Function Documentation . . . 160
12.21.2.1 is_dir . . . 160
12.21.2.2 is_exec . . . 160
12.21.2.3 is_file . . . 161
12.21.2.4 mk_dir . . . 161
12.22 Openldap/SASL Interface . . . 163
12.22.1 Detailed Description . . . 165
12.22.2 Typedef Documentation . . . 165
12.22.2.1 ldap_add . . . 165
12.22.2.2 ldap_conn . . . 165
12.22.2.3 ldap_modify . . . 165
12.22.3 Enumeration Type Documentation . . . 165
12.22.3.1 ldap_attrtype . . . 165
12.22.3.2 ldap_starttls . . . 165
12.22.4 Function Documentation . . . 166
12.22.4.1 ldap_add_attr . . . 166
12.22.4.2 ldap_addinit . . . 166
12.22.4.3 ldap_connect . . . 167
12.22.4.4 ldap_doadd . . . 168
12.22.4.5 ldap_domodify . . . 169
12.22.4.6 ldap_errmsg . . . 170
12.22.4.7 ldap_getattr . . . 171
12.22.4.8 ldap_getentry . . . 172
12.22.4.9 ldap_mod_add . . . 172
12.22.4.10 ldap_mod_addattr . . . 173
12.22.4.11 ldap_mod_del . . . 173
12.22.4.12 ldap_mod_delattr . . . 174

12.22.4.13 ldap_mod_remattr . . . 175
12.22.4.14 ldap_mod_rep . . . 175
12.22.4.15 ldap_mod_repattr . . . 175
12.22.4.16 ldap_modifyinit . . . 176
12.22.4.17 ldap_saslbind . . . 177
12.22.4.18 ldap_search_base . . . 178
12.22.4.19 ldap_search_one . . . 179
12.22.4.20 ldap_search_sub . . . 180
12.22.4.21 ldap_simplebind . . . 180
12.22.4.22 ldap_simplerebind . . . 181
12.22.4.23 ldap_unref_attr . . . 182
12.22.4.24 ldap_unref_entry . . . 183
12.23 XML Interface . . . 184
12.23.1 Detailed Description . . . 185
12.23.2 Typedef Documentation . . . 185
12.23.2.1 xml_doc . . . 185
12.23.2.2 xml_node . . . 186
12.23.2.3 xml_search . . . 186
12.23.3 Function Documentation . . . 186
12.23.3.1 xml_addnode . . . 186
12.23.3.2 xml_appendnode . . . 187
12.23.3.3 xml_close . . . 187
12.23.3.4 xml_createpath . . . 187
12.23.3.5 xml_delete . . . 189
12.23.3.6 xml_doctobuffer . . . 190
12.23.3.7 xml_free_buffer . . . 190
12.23.3.8 xml_getattr . . . 190
12.23.3.9 xml_getbuffer . . . 191
12.23.3.10 xml_getfirstnode . . . 191
12.23.3.11 xml_getnextnode . . . 192
12.23.3.12 xml_getnode . . . 193
12.23.3.13 xml_getnodes . . . 193
12.23.3.14 xml_getrootname . . . 193
12.23.3.15 xml_getrootnode . . . 194
12.23.3.16 xml_init . . . 194
12.23.3.17 xml_loadbuf . . . 194
12.23.3.18 xml_loaddoc . . . 195
12.23.3.19 xml_modify . . . 196
12.23.3.20 xml_nodecount . . . 197
12.23.3.21 xml_savefile . . . 197

12.23.3.22 xml_setattr . . . 198
12.23.3.23 xml_unlink . . . 198
12.23.3.24 xml_xpath . . . 198
12.24 XSLT Interface . . . 200
12.24.1 Detailed Description . . . 200
12.24.2 Typedef Documentation . . . 200
12.24.2.1 xslt_doc . . . 200
12.24.3 Function Documentation . . . 201
12.24.3.1 xslt_addparam . . . 201
12.24.3.2 xslt_apply . . . 202
12.24.3.3 xslt_apply_buffer . . . 203
12.24.3.4 xslt_clearparam . . . 204
12.24.3.5 xslt_close . . . 204
12.24.3.6 xslt_init . . . 205
12.24.3.7 xslt_open . . . 205
12.25 CURL Url interface. . . . 206
12.25.1 Detailed Description . . . 207
12.25.2 Typedef Documentation . . . 207
12.25.2.1 curl_authcb . . . 207
12.25.2.2 curl_post . . . 207
12.25.2.3 curl_progress_func . . . 207
12.25.2.4 curl_progress_newdata . . . 208
12.25.2.5 curl_progress_pause . . . 208
12.25.3 Function Documentation . . . 208
12.25.3.1 curl_buf2xml . . . 208
12.25.3.2 curl_geturl . . . 209
12.25.3.3 curl_newauth . . . 209
12.25.3.4 curl_newpost . . . 210
12.25.3.5 curl_postitem . . . 210
12.25.3.6 curl_posturl . . . 211
12.25.3.7 curl_setauth_cb . . . 211
12.25.3.8 curl_setprogress . . . 212
12.25.3.9 curl_ungzip . . . 212
12.25.3.10 curlclose . . . 213
12.25.3.11 curlinit . . . 213
12.25.3.12 url_escape . . . 214
12.25.3.13 url_unescape . . . 214
12.26 Zlib Interface . . . 216
12.26.1 Detailed Description . . . 216
12.26.2 Function Documentation . . . 216

12.26.2.1 gzinflatebuf . . . 216
12.26.2.2 is_gzip . . . 217
12.26.2.3 zcompress . . . 217
12.26.2.4 zuncompress . . . 218
12.27 Burtle Bob hash algorythim. . . . 219
12.27.1 Detailed Description . . . 219
12.27.2 Macro Definition Documentation . . . 220
12.27.2.1 final . . . 220
12.27.2.2 HASH_BIG_ENDIAN . . . 221
12.27.2.3 HASH_LITTLE_ENDIAN . . . 221
12.27.2.4 hashmask . . . 221
12.27.2.5 hashsize . . . 221
12.27.2.6 jenhash . . . 221
12.27.2.7 JHASH_INITVAL . . . 221
12.27.2.8 mix . . . 221
12.27.2.9 rot . . . 222
12.27.3 Function Documentation . . . 222
12.27.3.1 hashbig . . . 222
12.27.3.2 hashlittle . . . 225
12.27.3.3 hashlittle2 . . . 228
12.27.3.4 hashword . . . 232
12.27.3.5 hashword2 . . . 232
12.28 IPv6 Nat Mapping . . . 234
12.28.1 Detailed Description . . . 234
12.28.2 Typedef Documentation . . . 234
12.28.2.1 natmap . . . 234
12.28.3 Function Documentation . . . 234
12.28.3.1 rfc6296_map . . . 234
12.28.3.2 rfc6296_map_add . . . 235
12.28.3.3 rfc6296_test . . . 236
12.29 Windows Support . . . 238
12.29.1 Detailed Description . . . 238
12.29.2 Function Documentation . . . 238
12.29.2.1 get_ifinfo . . . 238
12.29.2.2 inet_ntop . . . 239
12.30 Distrotech Application Library (Todo) . . . 241
12.30.1 Detailed Description . . . 241
12.31 Linux Netfilter . . . 242
12.31.1 Detailed Description . . . 242
12.32 Connection Tracking . . . 243

12.32.1 Detailed Description . . . 243
12.32.2 Typedef Documentation . . . 243
12.32.2.1 nfct_struct . . . 243
12.32.3 Enumeration Type Documentation . . . 244
12.32.3.1 NF_CTRACK_FLAGS . . . 244
12.32.4 Function Documentation . . . 244
12.32.4.1 nf_ctrack_buildct . . . 244
12.32.4.2 nf_ctrack_close . . . 244
12.32.4.3 nf_ctrack_delete . . . 245
12.32.4.4 nf_ctrack_dump . . . 245
12.32.4.5 nf_ctrack_endtrace . . . 245
12.32.4.6 nf_ctrack_init . . . 246
12.32.4.7 nf_ctrack_nat . . . 246
12.32.4.8 nf_ctrack_trace . . . 246
12.33 Queue interface . . . 248
12.33.1 Detailed Description . . . 248
12.33.2 Typedef Documentation . . . 248
12.33.2.1 nfq_data . . . 248
12.33.2.2 nfq_queue . . . 249
12.33.2.3 nfqnl_msg_packet_hdr . . . 249
12.33.2.4 nfqueue_cb . . . 249
12.33.3 Enumeration Type Documentation . . . 249
12.33.3.1 NF_QUEUE_FLAGS . . . 249
12.33.4 Function Documentation . . . 249
12.33.4.1 nfqueue_attach . . . 249
12.33.4.2 snprintf_pkt . . . 250

13 Data Structure Documentation **253**

13.1 basic_auth Struct Reference . . . 253
13.1.1 Detailed Description . . . 253
13.1.2 Field Documentation . . . 253
13.1.2.1 passwd . . . 253
13.1.2.2 user . . . 253
13.2 blist_obj Struct Reference . . . 254
13.2.1 Detailed Description . . . 254
13.2.2 Field Documentation . . . 254
13.2.2.1 data . . . 254
13.2.2.2 hash . . . 254
13.2.2.3 next . . . 254
13.2.2.4 prev . . . 255

13.3 bucket_list Struct Reference . . . 255
13.3.1 Detailed Description . . . 255
13.3.2 Field Documentation . . . 255
13.3.2.1 bucketbits . . . 255
13.3.2.2 count . . . 255
13.3.2.3 hash_func . . . 256
13.3.2.4 list . . . 256
13.3.2.5 locks . . . 256
13.3.2.6 version . . . 256
13.4 bucket_loop Struct Reference . . . 256
13.4.1 Detailed Description . . . 257
13.4.2 Field Documentation . . . 257
13.4.2.1 blist . . . 257
13.4.2.2 bucket . . . 257
13.4.2.3 cur . . . 257
13.4.2.4 cur_hash . . . 257
13.4.2.5 head . . . 257
13.4.2.6 head_hash . . . 257
13.4.2.7 version . . . 258
13.5 config_category Struct Reference . . . 258
13.5.1 Detailed Description . . . 258
13.5.2 Field Documentation . . . 258
13.5.2.1 entries . . . 258
13.5.2.2 name . . . 258
13.6 config_entry Struct Reference . . . 258
13.6.1 Detailed Description . . . 259
13.6.2 Field Documentation . . . 259
13.6.2.1 item . . . 259
13.6.2.2 value . . . 259
13.7 config_file Struct Reference . . . 259
13.7.1 Detailed Description . . . 259
13.7.2 Field Documentation . . . 259
13.7.2.1 cat . . . 259
13.7.2.2 filename . . . 260
13.7.2.3 filepath . . . 260
13.8 curl_post Struct Reference . . . 260
13.8.1 Detailed Description . . . 260
13.8.2 Field Documentation . . . 260
13.8.2.1 first . . . 260
13.8.2.2 last . . . 260

13.9 curlbuf Struct Reference . . . 261
13.9.1 Detailed Description . . . 261
13.9.2 Field Documentation . . . 261
13.9.2.1 body . . . 261
13.9.2.2 bsize . . . 261
13.9.2.3 c_type . . . 261
13.9.2.4 header . . . 262
13.9.2.5 hsize . . . 262
13.10 framework_core Struct Reference . . . 262
13.10.1 Detailed Description . . . 262
13.10.2 Field Documentation . . . 263
13.10.2.1 developer . . . 263
13.10.2.2 email . . . 263
13.10.2.3 flags . . . 263
13.10.2.4 flock . . . 263
13.10.2.5 progname . . . 263
13.10.2.6 runfile . . . 263
13.10.2.7 sa . . . 263
13.10.2.8 sig_handler . . . 264
13.10.2.9 www . . . 264
13.10.2.10 year . . . 264
13.11 fwsocket Struct Reference . . . 264
13.11.1 Detailed Description . . . 265
13.11.2 Field Documentation . . . 265
13.11.2.1 addr . . . 265
13.11.2.2 children . . . 265
13.11.2.3 flags . . . 265
13.11.2.4 parent . . . 265
13.11.2.5 proto . . . 265
13.11.2.6 sock . . . 266
13.11.2.7 ssl . . . 266
13.11.2.8 type . . . 266
13.12 ifinfo Struct Reference . . . 266
13.12.1 Detailed Description . . . 267
13.12.2 Field Documentation . . . 267
13.12.2.1 idx . . . 267
13.12.2.2 ifaddr . . . 267
13.12.2.3 ipv4addr . . . 267
13.12.2.4 ipv6addr . . . 267
13.13 ipaddr_req Struct Reference . . . 267

13.13.1 Detailed Description . . . 268
13.13.2 Field Documentation . . . 268
13.13.2.1 buf . . . 268
13.13.2.2 i . . . 268
13.13.2.3 n . . . 268
13.14 iplink_req Struct Reference . . . 268
13.14.1 Detailed Description . . . 269
13.14.2 Field Documentation . . . 269
13.14.2.1 buf . . . 269
13.14.2.2 i . . . 269
13.14.2.3 n . . . 269
13.15 ldap_add Struct Reference . . . 269
13.15.1 Detailed Description . . . 269
13.15.2 Field Documentation . . . 270
13.15.2.1 bl . . . 270
13.15.2.2 dn . . . 270
13.16 ldap_attr Struct Reference . . . 270
13.16.1 Detailed Description . . . 270
13.16.2 Field Documentation . . . 270
13.16.2.1 count . . . 270
13.16.2.2 name . . . 271
13.16.2.3 next . . . 271
13.16.2.4 prev . . . 271
13.16.2.5 vals . . . 271
13.17 ldap_attrval Struct Reference . . . 271
13.17.1 Detailed Description . . . 271
13.17.2 Field Documentation . . . 272
13.17.2.1 buffer . . . 272
13.17.2.2 len . . . 272
13.17.2.3 type . . . 272
13.18 ldap_conn Struct Reference . . . 272
13.18.1 Detailed Description . . . 272
13.18.2 Field Documentation . . . 273
13.18.2.1 ldap . . . 273
13.18.2.2 limit . . . 273
13.18.2.3 sasl . . . 273
13.18.2.4 sctrlsp . . . 273
13.18.2.5 simple . . . 273
13.18.2.6 timelim . . . 273
13.18.2.7 uri . . . 273

13.19ldap_entry Struct Reference . . . 274
13.19.1 Detailed Description . . . 274
13.19.2 Field Documentation . . . 274
13.19.2.1 attrs . . . 274
13.19.2.2 dn . . . 274
13.19.2.3 dnufn . . . 274
13.19.2.4 first_attr . . . 275
13.19.2.5 list . . . 275
13.19.2.6 next . . . 275
13.19.2.7 prev . . . 275
13.19.2.8 rdn . . . 275
13.19.2.9 rdncnt . . . 275
13.20ldap_modify Struct Reference . . . 275
13.20.1 Detailed Description . . . 276
13.20.2 Field Documentation . . . 276
13.20.2.1 bl . . . 276
13.20.2.2 dn . . . 276
13.21ldap_modreq Struct Reference . . . 276
13.21.1 Detailed Description . . . 276
13.21.2 Field Documentation . . . 276
13.21.2.1 attr . . . 276
13.21.2.2 cnt . . . 277
13.21.2.3 first . . . 277
13.21.2.4 last . . . 277
13.22ldap_modval Struct Reference . . . 277
13.22.1 Detailed Description . . . 277
13.22.2 Field Documentation . . . 277
13.22.2.1 next . . . 277
13.22.2.2 value . . . 277
13.23ldap_rdn Struct Reference . . . 278
13.23.1 Detailed Description . . . 278
13.23.2 Field Documentation . . . 278
13.23.2.1 name . . . 278
13.23.2.2 next . . . 278
13.23.2.3 prev . . . 278
13.23.2.4 value . . . 278
13.24ldap_results Struct Reference . . . 279
13.24.1 Detailed Description . . . 279
13.24.2 Field Documentation . . . 279
13.24.2.1 count . . . 279

13.24.2.2 entries . . . 279
13.24.2.3 first_entry . . . 279
13.25ldap_simple Struct Reference . . . 279
13.25.1 Detailed Description . . . 280
13.25.2 Field Documentation . . . 280
13.25.2.1 cred . . . 280
13.25.2.2 dn . . . 280
13.26natmap Struct Reference . . . 280
13.26.1 Detailed Description . . . 281
13.26.2 Field Documentation . . . 281
13.26.2.1 adji . . . 281
13.26.2.2 adjo . . . 281
13.26.2.3 epre . . . 281
13.26.2.4 ipre . . . 281
13.26.2.5 mask . . . 281
13.27nfq_queue Struct Reference . . . 281
13.27.1 Detailed Description . . . 282
13.27.2 Field Documentation . . . 282
13.27.2.1 cb . . . 282
13.27.2.2 data . . . 282
13.27.2.3 nfq . . . 282
13.27.2.4 num . . . 282
13.27.2.5 qh . . . 282
13.28nfq_struct Struct Reference . . . 282
13.28.1 Detailed Description . . . 283
13.28.2 Field Documentation . . . 283
13.28.2.1 fd . . . 283
13.28.2.2 flags . . . 283
13.28.2.3 h . . . 283
13.28.2.4 pf . . . 283
13.29pseudohdr Struct Reference . . . 283
13.29.1 Detailed Description . . . 283
13.29.2 Field Documentation . . . 284
13.29.2.1 daddr . . . 284
13.29.2.2 len . . . 284
13.29.2.3 proto . . . 284
13.29.2.4 saddr . . . 284
13.29.2.5 zero . . . 284
13.30radius_connection Struct Reference . . . 284
13.30.1 Detailed Description . . . 285

13.30.2 Field Documentation . . . 285
13.30.2.1 id . . . 285
13.30.2.2 server . . . 285
13.30.2.3 sessions . . . 285
13.30.2.4 socket . . . 285
13.31 radius_packet Struct Reference . . . 285
13.31.1 Detailed Description . . . 286
13.31.2 Field Documentation . . . 286
13.31.2.1 attrs . . . 286
13.31.2.2 code . . . 286
13.31.2.3 id . . . 286
13.31.2.4 len . . . 286
13.31.2.5 token . . . 286
13.32 radius_server Struct Reference . . . 287
13.32.1 Detailed Description . . . 287
13.32.2 Field Documentation . . . 287
13.32.2.1 acctport . . . 287
13.32.2.2 authport . . . 287
13.32.2.3 connex . . . 287
13.32.2.4 id . . . 288
13.32.2.5 name . . . 288
13.32.2.6 secret . . . 288
13.32.2.7 service . . . 288
13.32.2.8 timeout . . . 288
13.33 radius_session Struct Reference . . . 288
13.33.1 Detailed Description . . . 289
13.33.2 Field Documentation . . . 289
13.33.2.1 cb_data . . . 289
13.33.2.2 id . . . 289
13.33.2.3 minserver . . . 289
13.33.2.4 olen . . . 289
13.33.2.5 packet . . . 289
13.33.2.6 passwd . . . 290
13.33.2.7 read_cb . . . 290
13.33.2.8 request . . . 290
13.33.2.9 retries . . . 290
13.33.2.10 sent . . . 290
13.34 ref_obj Struct Reference . . . 290
13.34.1 Detailed Description . . . 291
13.34.2 Field Documentation . . . 291

13.34.2.1 cnt . . . 291
13.34.2.2 data . . . 291
13.34.2.3 destroy . . . 291
13.34.2.4 lock . . . 291
13.34.2.5 magic . . . 291
13.34.2.6 size . . . 291
13.35 sasl_defaults Struct Reference . . . 292
13.35.1 Detailed Description . . . 292
13.35.2 Field Documentation . . . 292
13.35.2.1 authcid . . . 292
13.35.2.2 authzid . . . 292
13.35.2.3 mech . . . 293
13.35.2.4 passwd . . . 293
13.35.2.5 realm . . . 293
13.36 socket_handler Struct Reference . . . 293
13.36.1 Detailed Description . . . 293
13.36.2 Field Documentation . . . 294
13.36.2.1 cleanup . . . 294
13.36.2.2 client . . . 294
13.36.2.3 connect . . . 294
13.36.2.4 data . . . 294
13.36.2.5 sock . . . 294
13.37 sockstruct Union Reference . . . 294
13.37.1 Detailed Description . . . 295
13.37.2 Field Documentation . . . 295
13.37.2.1 sa . . . 295
13.37.2.2 sa4 . . . 295
13.37.2.3 sa6 . . . 295
13.37.2.4 ss . . . 295
13.37.2.5 un . . . 295
13.38 ssldata Struct Reference . . . 295
13.38.1 Detailed Description . . . 296
13.38.2 Field Documentation . . . 296
13.38.2.1 bio . . . 296
13.38.2.2 ctx . . . 296
13.38.2.3 flags . . . 296
13.38.2.4 meth . . . 296
13.38.2.5 parent . . . 297
13.38.2.6 ssl . . . 297
13.39 thread_pvt Struct Reference . . . 297

13.39.1 Detailed Description . . . 297
13.39.2 Field Documentation . . . 297
13.39.2.1 cleanup . . . 297
13.39.2.2 data . . . 298
13.39.2.3 flags . . . 298
13.39.2.4 func . . . 298
13.39.2.5 sighandler . . . 298
13.39.2.6 thr . . . 298
13.40 threadcontainer Struct Reference . . . 298
13.40.1 Detailed Description . . . 299
13.40.2 Field Documentation . . . 299
13.40.2.1 list . . . 299
13.40.2.2 manager . . . 299
13.41 unixclient_sockthread Struct Reference . . . 299
13.41.1 Detailed Description . . . 299
13.41.2 Field Documentation . . . 300
13.41.2.1 client . . . 300
13.41.2.2 data . . . 300
13.41.2.3 endpoint . . . 300
13.41.2.4 sock . . . 300
13.42 unixserv_sockthread Struct Reference . . . 300
13.42.1 Detailed Description . . . 301
13.42.2 Field Documentation . . . 301
13.42.2.1 data . . . 301
13.42.2.2 mask . . . 301
13.42.2.3 protocol . . . 301
13.42.2.4 read . . . 301
13.42.2.5 sock . . . 301
13.42.2.6 sockpath . . . 301
13.43 xml_attr Struct Reference . . . 302
13.43.1 Detailed Description . . . 302
13.43.2 Field Documentation . . . 302
13.43.2.1 name . . . 302
13.43.2.2 value . . . 302
13.44 xml_node Struct Reference . . . 302
13.44.1 Detailed Description . . . 303
13.44.2 Field Documentation . . . 303
13.44.2.1 attrs . . . 303
13.44.2.2 key . . . 303
13.44.2.3 name . . . 303

13.44.2.4 nodeptr . . . 303
13.44.2.5 value . . . 303
13.45 xml_node_iter Struct Reference . . . 304
13.45.1 Detailed Description . . . 304
13.45.2 Field Documentation . . . 304
13.45.2.1 cnt . . . 304
13.45.2.2 curpos . . . 304
13.45.2.3 xsearch . . . 304
13.46 xml_search Struct Reference . . . 304
13.46.1 Detailed Description . . . 305
13.46.2 Field Documentation . . . 305
13.46.2.1 nodes . . . 305
13.46.2.2 xmldoc . . . 305
13.46.2.3 xpathObj . . . 305
13.47 xslt_doc Struct Reference . . . 305
13.47.1 Detailed Description . . . 306
13.47.2 Field Documentation . . . 306
13.47.2.1 doc . . . 306
13.47.2.2 params . . . 306
13.48 xslt_param Struct Reference . . . 306
13.48.1 Detailed Description . . . 306
13.48.2 Field Documentation . . . 307
13.48.2.1 name . . . 307
13.48.2.2 value . . . 307
13.49 zobj Struct Reference . . . 307
13.49.1 Detailed Description . . . 307
13.49.2 Field Documentation . . . 307
13.49.2.1 buff . . . 307
13.49.2.2 olen . . . 308
13.49.2.3 zlen . . . 308

14 File Documentation **309**

14.1 doxygen/dox/buckets.dox File Reference . . . 309
14.2 doxygen/dox/examples.dox File Reference . . . 309
14.3 doxygen/dox/index.dox File Reference . . . 309
14.4 doxygen/dox/main.dox File Reference . . . 309
14.5 doxygen/dox/modules.dox File Reference . . . 309
14.6 doxygen/dox/refobj.dox File Reference . . . 309
14.7 doxygen/dox/sockets.dox File Reference . . . 309
14.8 doxygen/dox/sockex.dox File Reference . . . 309

14.9 doxygen/dox/thread.dox File Reference . . . 309
14.10 doxygen/examples/socket.c File Reference . . . 309
14.10.1 Detailed Description . . . 310
14.10.2 Function Documentation . . . 310
14.10.2.1 accept_func . . . 310
14.10.2.2 client_func . . . 310
14.10.2.3 FRAMEWORK_MAIN . . . 311
14.10.2.4 server_func . . . 312
14.10.2.5 socktest . . . 312
14.10.2.6 unixsocktest . . . 313
14.11 socket.c . . . 314
14.12 src/socket.c File Reference . . . 316
14.12.1 Detailed Description . . . 317
14.13 socket.c . . . 317
14.14 src/config.c File Reference . . . 324
14.14.1 Detailed Description . . . 325
14.15 config.c . . . 325
14.16 src/curl.c File Reference . . . 329
14.16.1 Detailed Description . . . 330
14.17 curl.c . . . 330
14.18 src/fileutil.c File Reference . . . 335
14.18.1 Detailed Description . . . 336
14.19 fileutil.c . . . 336
14.20 src/include/dtsapp.h File Reference . . . 337
14.20.1 Detailed Description . . . 350
14.21 dtsapp.h . . . 350
14.22 src/interface.c File Reference . . . 358
14.22.1 Detailed Description . . . 360
14.23 interface.c . . . 360
14.24 src/iputil.c File Reference . . . 369
14.24.1 Detailed Description . . . 370
14.25 iputil.c . . . 370
14.26 src/libxml2.c File Reference . . . 375
14.26.1 Detailed Description . . . 377
14.27 libxml2.c . . . 377
14.28 src/libxslt.c File Reference . . . 385
14.28.1 Detailed Description . . . 386
14.29 libxslt.c . . . 386
14.30 src/lookup3.c File Reference . . . 388
14.30.1 Detailed Description . . . 389

14.31 lookup3.c . . . 389
14.32 src/main.c File Reference . . . 402
14.32.1 Detailed Description . . . 403
14.33 main.c . . . 403
14.34 src/nf_ctrack.c File Reference . . . 406
14.34.1 Detailed Description . . . 407
14.35 nf_ctrack.c . . . 407
14.36 src/nf_queue.c File Reference . . . 410
14.36.1 Detailed Description . . . 411
14.37 nf_queue.c . . . 411
14.38 src/openldap.c File Reference . . . 415
14.38.1 Detailed Description . . . 417
14.39 openldap.c . . . 417
14.40 src/radius.c File Reference . . . 434
14.40.1 Detailed Description . . . 435
14.41 radius.c . . . 436
14.42 src/refobj.c File Reference . . . 442
14.42.1 Detailed Description . . . 444
14.43 refobj.c . . . 444
14.44 src/rfc6296.c File Reference . . . 451
14.44.1 Detailed Description . . . 451
14.45 rfc6296.c . . . 452
14.46 src/sslutil.c File Reference . . . 454
14.46.1 Detailed Description . . . 455
14.47 sslutil.c . . . 455
14.48 src/thread.c File Reference . . . 464
14.48.1 Detailed Description . . . 465
14.49 thread.c . . . 465
14.50 src/unixsock.c File Reference . . . 470
14.50.1 Detailed Description . . . 470
14.51 unixsock.c . . . 471
14.52 src/util.c File Reference . . . 475
14.52.1 Detailed Description . . . 476
14.53 util.c . . . 477
14.54 src/winiface.cpp File Reference . . . 481
14.54.1 Detailed Description . . . 481
14.55 winiface.cpp . . . 481
14.56 src/zlib.c File Reference . . . 483
14.56.1 Detailed Description . . . 483
14.57 zlib.c . . . 483

15 Example Documentation **487**

15.1 socket.c 487

Index **489**

Chapter 1

Distrotech Application Library Manual

1.1 Introduction

This library has grown over time to include various interfaces as i have required them or have experimented with them and added them.

The core functionality are the referenced lockable data structures that i reimplemented for my own experimentation to understand how and why they were used in the asterisk project.

There is now a partner to this lib to support GUI applications using the wxWidgets library.

1.2 Further information

Please see the following links before diving into the modules.

Application Startup

Referenced Lockable Objects

Hashed Bucket Lists

Thread Interface

Socket Interface

Todo List

1.3 Copyright information.

Author

Gregory Nietsky [Distrotech Solutions] <gregory@distrotech.co.za>

Date

2010-

Chapter 2

Application Startup

2.1 Using helper macro instead of main()

This library includes functions to simplify startup.

- printgnu() Displays a standard message on the console at startup.
- daemonize() Forks and exits the process to run it in the background.
- lockpidfile() Creates a file that contains the pid and locks it.
- seedrand() Seed the random number generator.
- sslstartup() Start open ssl.
- Install a default signal handler and use a callback to handle signals (Not supported on WIN32).

These are all wrapped up in a macro FRAMEWORK_MAIN() that replaces main(). This is done by implementing main creating a callback initialising the services and calling the callback

```
FRAMEWORK_MAIN("Socket Client/Server Echo (TCP/TLS/UDP/DTLS)", "Gregory Hinton Nietsky", "
      gregory@distrotech.co.za",
        "http://www.distrotech.co.za", 2013, "/var/run/sockettest",
      FRAMEWORK_FLAG_DAEMONLOCK, NULL) {

    if (argc < 3) {
#ifndef __WIN32
        printf("Requires arguments %s [tcp|tls|udp|dtls|unix_d|unix_s] [ipaddr|socket]\n", argv[0]);
#else
        printf("Requires arguments %s [tcp|tls|udp|dtls] ipaddr\n", argv[0]);
#endif
        return (-1);
    }

    daemonize();
```

As you can see this macro has replaced main() you have access to the arg count and arg list as usual via argc / argv.

Various flags control the behaviour in this case daemonize was run after the args have been checked.

See Also

framework_flags
framework_init()
framework_mkcore()

Warning

memory allocated by framework_mkcore() is only released by framework_init() always call framework_mkcore() first and always call framework_init() when calling framework_mkcore().

Chapter 3

Referenced Lockable Objects

3.1 Introduction

Data structures in C are are simple by nature well defined and logical. Using pointers is almost imperitive for performance issues copying chunks of data onto the limited space on the stack is not the best solution.

For these same reasons you would use dynamically allocated memory (malloc/calloc) and assign it to your struct, a very important reason not to do it on the stack is that when you leave the function the stack space is feeed.

So assuming all data structs are pointers the only access to the data is via these pointers the pointers can be copied and overwriten as needed the memory is available till freed.

take the following code into account

```
struct cust *c1, *c2, *c3, c4;

c1 = malloc(sizeof(struct cust));
c2 = malloc(sizeof(struct cust));
c3 = malloc(sizeof(struct cust));

.... assign the data ....

if (c1->priority < c3->priority) {
  c4 = c1;
  c1 = c3;
  c3 = c4;
}
```

It is clear that its possible that c4 and c3 are pointing to the identical memory here is where the problem starts if i free c4 and then try access c3 its possible that the data will be corrupted or reassigned causeing unpredictable results.

This is the first problem referenced objects solve that no memory will be freed while a referece is held for the object. use of objalloc() instead of malloc/calloc will return a pointer to the allocated memory just as before but now when we want to copy the pointer and ensure it persists we can refrence it using objref() and release the refrence with objunref() if the refrence count is 0 the object will be freed the current count is returned by objcnt().

lets look at the code again but using refereneced objects of course if c4 is a tmp variable that wont change and be used again there is no need to do this we assuming that this is uncertain and taking precautions.

```
struct cust *c1, *c2, *c3, c4;

c1 = objalloc(sizeof(strict cust), NULL);
c2 = objalloc(sizeof(struct cust), NULL);
c3 = objalloc(sizeof(struct cust), NULL);

.... assign the data ....

if (c1->priority < c3->priority) {
```

```
    /*grab a new ref for c1 and pass to c4*/
    c4 = (objref(c1)) ? c1 : NULL;

    /*grab ref for c3 and pass to c1*/;
    c1 = (objref(c3)) ? c3 : NULL;

    /* pass the ref of c4 to c3*/
    c3 = c4;
    /* release the reference for old c3 now c1*/
    objunref(c1);

    /* we now have 2 refs to c3 the original c1 and one ref for the others.*/
  }
```

The second parameter of objalloc() is the "destructor" this is a function callback to cleanup the data before it is freed by objunref.

This is a slightly pointless bit of code but you should notice that we have called objref 2 and objunref 1 you should also see that reference can be passed with the pointer. The original c1 is now referenced 2 once in c4 and once in c3. the reason we dont just call objref on c4 at the end is in multi threaded applications its possible to have things get scrambled and a item freed in another thread before you reference it its best to always call objref before copying the reference use of locking is needed in some circumstances. if you want to grab a reference to a shared memory location that is "changeable" locking is required.

this is done implicitly with objref() / objunref() the reference is obtained atomically the return value of objref should be checked it it is 0 then the referenced failed also to prevent a dead lock never call objref while holding the lock for the reference.

3.2 Other referenced object functions.

Referenced objects can be locked and unlocked but not reentrantly (this is a design choice and can be made optional). the functions objlock() will lock and objunlock() will unlock referenced objects a lock can be attempted using objtrylock().

The size of the requested memory is available by calling objsize() returning a new reference to a string is done with objchar().

The macros setflag clearflag and testflag for atomically handling flags.

3.3 Internal workings.

There is no voodo or black magic to the workings of a referenced object they are all ref_obj structures.

When objalloc() is called a a block of memmory the size requested + the size of ref_obj is allocatted and a pointer to data is returned and the data is set to the to the block after the ref_obj. when the objXXX() functions are called the pointer provided is rewound to the begining of the ref_obj the value of ref_obj::magic is checked to ensure that it is a referenced object and -1 is returned if it is not.

objlock() / objunlock() / objtrylock() will lock the mutex ref_obj::lock.

objref() / objunref() will first lock ref_obj::lock then alter ref_obj::cnt when the count reaches 0 the destructor callback ref_obj::destroy is called with ref_obj::data and on return the memory is freed. this is very similar to a C++ destructor.

objcnt() returns the value of ref_obj::cnt obtained while ref_obj::lock is held or -1 on error it is a error to return 0 as ref_obj::magic is set to zero when the count reaches 0.

objsize() returns ref_obj::size this contains the size of the memmory allocated (total).

3.4 Referenced Lockable Objects With Classes (C++)

C++ classes implement destructors but do not implement reference counting by overloading the new/delete operators it is possible to use referenced objects with C++ classes.

include the macro DTS_OJBREF_CLAS in your C++ class as follows. as it declares the destructor this does not need to be redeclared.

```
class somecool_class {
    public:
        DTS_OJBREF_CLASS(somecool_class);
...........
...........
}
```

The macro is included below internally it replaces new with objalloc and calls the cleanup routine it creates this calls delete that will run the destructor.

```
void *operator new(size_t sz) {\
    return objalloc(sz, &classtype::dts_unref_classtype);\
}\
void operator delete(void *obj) {\
}\
static void dts_unref_classtype(void *data) {\
    delete (classtype*)data;\
}\
~classtype()
```

Note

This should only be used when there is no inheritance.

3.5 Downsides

It adds ref_obj size memory to each referenced object this includes the size of the lock structure, however with almost all programs but the simplest benifiting from multi threading this is only a disadvantage in the simplest programs.

On a 32bit system 20bytes is used for ref_obj and on 64bit 32bytes is used excluding the size of the lock 24bytes and 40bytes respectivly, taaking into account the availability of memory and the benifits this will be acceptable.

One option is to drop support for objsize() this will save 4bytes and 8 bytes respectivly removing the magic cookie is not recomended.

Chapter 4

Hashed Bucket Lists

4.1 Introduction

The only method of creating the concept of a list is via an array these are not ideal and have the folowing downsides.

- The size of the array needs to be known ahead of time or it needs to be big enough and can be resized at a processing / memmory expense.
- Inserting a value at a position involves resizing as above then moving all of the existing items over one at a time.
- Removing elements from an array is either done in reverse to inserting or the item is NULL'd and left.
- Requires locking the whole array in multithreaded applications not only a record/records.

Arrays have the following upsides

- Elements can be accessed randomly if there position is known this is not so simple as the index is linear and can change on insertion/deletion.
- As the elements are adjacent in memory accessing them sequntially is faster than non seqential access.

The concept of the linked list was introduced to circumvent these downsides in its simplest form a structure will have a pointer of its data type called next initially this is null to add a element to the list you set the last element in the lists next pointer to the element adding while its next element is set to NULL

```
struct test {
    const char *name;
    struct test *next;
};


struct test a, b, c, *i;

a.name = "a";
a.next = NULL;

b.name = "b";
b.next = NULL;

c.name = "c";
c.next = NULL;

/*lets link em*/

a.next = &b;
b.next = &c;
```

```
for(i = &a; i; i=i->next) {
    .....
}
```

This shows the basic linked list and is effectivly equivilent as to interating through an array except for the loss of speed not been adjacent.

Double linked lists will have a prev pointer too that will allow traversal in any direction.

Looking at structure blist_obj you can see the next/prev pointers in addition to a hash and data pointer that points to the data so any item can be linked without the item itself requireing next/prev pointers this is the storage of all bucket lists.

The reason they are bucket lists is that they have elements of both arrays and linked lists the bucket list is infact a array of linked lists see the bucket_list structure.

The list is a array of 2^bucketbits these are the buckets so for 8192 elements using 6 bits will create 64 buckets if filled equally there will be 128 elements in each. this will allow better access to the chunk you want access too and allows for quicker more efficent traversal than traversing all 8192 elements.

Allocating the elements to a bucket is where the hash comes in each element will via some unique immutable "key" be hashed see jenhash() this hash will be masked with the bucket bits to determine the bucket to be placed in they are then inserted basesd on there hash, this allows the algorythim to search forward or backward theortically only ever having to traverse 64 elements of 8192.

Using this hybrid approach gives us a good compromise and benifits of either method.

The big disadvantage is that the data needs to have some immutable element to be able to search with and does not afford the same random access that arrays do but far better than standard linked lists. In both these cases with most data having some unique key and machines been faster with faster memory they acceptable.

4.2 Usage of hashed bucket lists

Hashed bucket lists are easy to use they are created with a call to create_bucketlist() you will need a hash function to generate the hash if you want to use search by key function you need to accept both the data and the key and return the hash.

```
int32_t hash(const void *data, int key) {
    int ret = 0;

    /*cast the data to the correct structure*/
    struct form_item *fi = data;
    /*Return the data as the key if we searching key=1 or the name otherwise*/
    const char *hashkey = (key) ? (const char*)data : fi->name;

    ret = jenhash(hashkey, strlen(hashkey), 0);

    return(ret);
}
```

Thats that folks use addtobucket() to add a item to the list, remove_bucket_item() to remove reference and bucket_list_cnt() to get number of elements in the list.

Searching the list can be done via iteration or by key using bucketlist_callback() and bucket_list_find_key() respectivly.

Too implement your own interator use init_bucket_loop() next_bucket_loop() and remove_bucket_loop().

Chapter 5

Thread Interface

5.1 Introduction

Most modern CPU's come with multiple cores the ability to thread a progam will allow taking advantage of these cores more fully. In a single core system using threads will allow processes to run in the background possibly waiting for input and "sleeping" this can happen while other processes continue.

This library makes use of threads on all sockets a socket is created and processed in its own thread.

The easiest way to see a thread is as a program inside a program a thread starts in a function with a reference to data supplied at thread initilization. what function is called and what data is provided is up to the programer.

On exiting the thread a cleanup function can be executed if required.

Its also possible on some systems [not windows] to handle signals that are delivered to the thread from the systems signal handler where it arrives in the thread. SIGUSR1 SIGUSR2 SIGHUP SIGALRM SIGINT and SIGTERM will be processed by thread signal handlers before been passed to the application handler.

A signal can be sent to a thread using pthread_kill external events are handled at application level.

5.2 Creating A Thread

A thread is created by calling framework_mkthread() passing the thread function, cleanup function, signal handler, refernece to data to pass to thread and options thread_option_flags.

By default NULL is returned and the thread is started not cancelable and detached its important to check framework-_threadok() periodically ideally as a loop control to check if the thread should exit shutdown will be blocked till all threads return unless they cacelable.

If the application is running under framework_init() or FRAMEWORK_MAIN() then on return of the "main" function stopthreads() is run . stopthreads() flags the manager thread for shutdown and terminate all runnig threads passing a non zerop value for the join paramater will cause the process to join and block on the management thead.

See Also

threadfunc
threadcleanup
threadsighandler
thread_option_flags

Chapter 6

Socket Interface

6.1 Introduction

The socket interface allows creating a thread per socket that passes output to a callback when available.

TCP/TLSv1/SSLv3/UDP are supported on linux and windows, additionaly SSLv2 is supported depending on the openssl implementation. DTLSv1 is supported on linux only.

Steps to creating a socket

- Create a SSL session if required use one of tlsv1_init() sslv2_init() sslv3_init() dtlsv1_init().
- Create a socket either as a server [bind] or client [connect] choices are tcpbind() tcpconnect() udpbind() udpconnect()
- Start up the client and or server threads using socketserver() and socketclient()
- When done call close_sock() on the socket.

6.2 SSL Support

Internally this is supplied from openssl various other openssl functions are used in this library ie base64 encoding. you will require a CA certificate[s] and a signed client certificate and key supply the paths to the initilization routines. The verify flag can be used to pass openssl verification flags.

Todo passphrase support

6.3 Socket Creation

To create a socket pass the socket creation function the ipaddr/hostname either ipv4 or ipv6 the port and the optional ssl session created above.

the result from this function will be the socket used in all other interactions.

6.4 Starting A Socket

A socket is started when the thread for the socket starts with socketclient or socketserver the latter creates a bucketlist for children and enables some extra options for DTLSv1.

They both requre the socket structure created above a callback routine called when data is available and a reference to data that is passed back in the callback. For thread management a thread cleanup function can be supplied that is called on thread closure this will allow cleaning up the data reference will be passed to this function as well.

In addition there is a optional callback for servers that will be called when a connection is accepted to allow for any handling needed on the server.

See Also

socketrecv
threadcleanup

6.5 Reading/Writeing To Sockets

There are 2 functions each for reading and writing to sockets socketread_d() and socketwrite_d() are required for stateless datagram sockets (UDP), they differ from socketread() and socketwrite() in that they use a additional addr paramater containing the remote address. passing NULL for this value is equivilent too socketread() / socketwrite().

6.6 Unix Domain Sockets

These are supported for SOCK_DGRAM and SOCK_STREAM and are capable of multiple connections.

unixsocket_server() and unixsocket_client() return sockets (fwsocket) and use of socketread_d() and socketwrite_d().

SOCK_DGRAM requires creating a tempoary socket file for use as a endpoint to support multiple connections this is handled internally but its best to only use SOCK_STREAM.

6.7 Multicast Sockets

Multicast sockets can be created and used as any other socket they only support SOCK_DGRAM traffic as there is no concept of client/server communication. A thread will be opened as a client writing to the socket should be done with socketwrite().

There 2 helper routines that allow generating multicast groups mcast4_ip() and mcast6_ip().

See Also

mcast_socket()
socketclient()

6.8 Example Code

Socket Example (Echo Server/Client) contains a example of socket code implementing a a echo server with 2 clients.

Chapter 7

Socket Example (Echo Server/Client)

7.1 Details

Application flow

- Check command line options if they correct daemonize.
- If required create SSL sessions.
- Create one server and 2 client sockets.
- Bind the server socket.
- Connect the clients to the server.
- Start the server thread.
- Start client threads.
- Write the client name to the server via the client socket.
- Echo back to the client and sleep the server thread for 1 second.
- Sleep the main thread for 5 seconds allowing exit.

7.2 Annotation

See doxygen/examples/socket.c for annotated source code.

7.3 Code

```
#ifdef __WIN32
#include <winsock2.h>
#include <stdint.h>
#else
#include <fcntl.h>
#endif

#include <string.h>
#include <stdio.h>

#include <openssl/ssl.h>
#include <dtsapp.h>

void accept_func(struct fwsocket *sock, void *data) {
}

void server_func(struct fwsocket *sock, void *data) {
```

```
    char buff[128];
    union sockstruct addr;

    if (socketread_d(sock, &buff, 128, &addr) > 0) {
        socketwrite_d(sock, &buff, strlen(buff) + 1, &addr);
        printf("[S] %s %i\n", buff, sock->sock);
        sleep(1);
    }
}

void client_func(struct fwsocket *sock, void *data) {
    char buff[128];

    if (socketread(sock, &buff, 128) > 0) {
        socketwrite(sock, &buff, strlen(buff) + 1);
        printf("[C] %s %i\n", buff, sock->sock);
    }
}

void socktest(const char *ipaddr, int tcp, int ssl) {
    struct fwsocket *serv, *client, *client2;
    void *ssl_c = NULL, *ssl_s = NULL, *ssl_c2 = NULL;
    char *buff = "client 1";
    char *buff2 = "client 2";
    int cnt;

    if (ssl && tcp) {
        ssl_s = sslv3_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
      SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
        ssl_c = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
        ssl_c2 = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
    } else if (ssl) {
        ssl_s = dtlsv1_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
     SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
        ssl_c = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
     SSL_VERIFY_NONE);
        ssl_c2 = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem"
      , SSL_VERIFY_NONE);
    }

    if (tcp) {
        serv = tcpbind(ipaddr, "1111", ssl_s, 10);
        client = tcpconnect(ipaddr, "1111", ssl_c);
        client2 = tcpconnect(ipaddr, "1111", ssl_c2);
    } else {
        serv = udpbind(ipaddr, "1111", ssl_s);
        client = udpconnect(ipaddr, "1111", ssl_c);
        client2 = udpconnect(ipaddr, "1111", ssl_c2);
    }

    if (serv && client && client2) {
        socketserver(serv, server_func, accept_func, NULL, NULL);
        socketclient(client, NULL, client_func, NULL);
        socketclient(client2, NULL, client_func, NULL);

        socketwrite(client, buff, strlen(buff)+1);
        socketwrite(client2, buff2, strlen(buff2)+1);

        sleep(5);
    } else {
        printf("ERROR\n");
    }

    close_socket(client);
    close_socket(client2);
    close_socket(serv);
}

#ifndef __WIN32
void unixsocktest(const char *socket, int protocol) {
    char *buff = "client 1";
    char *buff2 = "client 2";
    struct fwsocket *client, *client2, *server;

    server = unixsocket_server(socket, protocol, S_IXUSR | S_IWGRP | S_IRGRP | S_IXGRP |
      S_IWOTH | S_IROTH | S_IXOTH, server_func, NULL);
    sleep(1); /*wait for socket*/
    client = unixsocket_client(socket, protocol, client_func, NULL);
    client2 = unixsocket_client(socket, protocol, client_func, NULL);

    socketwrite_d(client, buff, strlen(buff)+1, NULL);
    socketwrite_d(client2, buff2, strlen(buff2)+1, NULL);

    sleep(5);
```

```
    close_socket(client);
    close_socket(client2);
    close_socket(server);
}
#endif

FRAMEWORK_MAIN("Socket Client/Server Echo (TCP/TLS/UDP/DTLS)", "Gregory Hinton Nietsky", "
      gregory@distrotech.co.za",
        "http://www.distrotech.co.za", 2013, "/var/run/sockettest",
      FRAMEWORK_FLAG_DAEMONLOCK, NULL) {

    if (argc < 3) {
#ifndef __WIN32
        printf("Requires arguments %s [tcp|tls|udp|dtls|unix_d|unix_s] [ipaddr|socket]\n", argv[0]);
#else
        printf("Requires arguments %s [tcp|tls|udp|dtls] ipaddr\n", argv[0]);
#endif
        return (-1);
    }

    daemonize();
    if (!strcmp(argv[1], "udp")) {
        socktest(argv[2], 0, 0);
    } else if (!strcmp(argv[1], "dtls")) {
        socktest(argv[2], 0, 1);
    } else if (!strcmp(argv[1], "tcp")) {
        socktest(argv[2], 1, 0);
    } else if (!strcmp(argv[1], "tls")) {
        socktest(argv[2], 1, 1);
#ifndef __WIN32
    } else if (!strcmp(argv[1], "unix_d")) {
        unixsocktest(argv[2], SOCK_DGRAM);
    } else if (!strcmp(argv[1], "unix_s")) {
        unixsocktest(argv[2], SOCK_STREAM);
#endif
    } else {
        printf("Invalid Option\n");
    }
}
```

7.4 Output

```
./socket tls ::1
Socket Client/Server Echo (TCP/TLS/UDP/DTLS)

Copyright (C) 2013 Gregory Hinton Nietsky <gregory@distrotech.co.za>

        http://www.distrotech.co.za

    This program comes with ABSOLUTELY NO WARRANTY
    This is free software, and you are welcome to redistribute it
    under certain conditions.

......./dtsapplib/private$ [S] client 1 19
[C] client 1 17
[S] client 2 20
[C] client 2 18
[S] client 1 19
[C] client 1 17
[S] client 2 20
[C] client 2 18
[S] client 1 19
[C] client 1 17
[S] client 2 20
[C] client 2 18
[S] client 1 19
[C] client 1 17
[S] client 2 20
[C] client 2 18
[S] client 1 19
[C] client 1 17
[S] client 2 20
[C] client 2 18
[S] client 1 19
[C] client 1 17
[S] client 2 20
[C] client 2 18
```

Chapter 8

Todo List

Global daemonize ()

WIN32 options is there a alternative for this.

Global framework_mkcore (char ∗progname, char ∗name, char ∗email, char ∗web, int year, char ∗runfile, int flags, syssighandler sigfunc)

does threads actually work in windows with no sighandler.

Global get_ifipaddr (const char ∗iface, int family)

WIN32 Support

Global get_ip6_addrprefix (const char ∗iface, unsigned char ∗prefix)

WIN32 support

Group LIB-OBJ-Bucket

Dont hash the memory supply a key perhaps a key array type.

Global mcast_socket (const char ∗iface, int family, const char ∗mcastip, const char ∗port, int flags)

Win32 support for inet_ntop/inet_pton

Global seedrand (void)

This wont work on WIN32

Global socketwrite_d (struct fwsocket ∗sock, const void ∗buf, int num, union sockstruct ∗addr)

implement send/sendto in WIN32

Global ssl_shutdown (void ∗data, int sock)

Make sure this is only called when the thread has stoped selecting here may be wrong.

Global touch (const char ∗filename, uid_t user, gid_t group)

WIN32 does not use uid/gid and move to file utils module.

Global xml_getfirstnode (struct xml_search ∗xpsearch, void ∗∗iter)

Thread safety when XML doc changes.

Global zuncompress (struct zobj ∗buff, uint8_t ∗obuff)

Implement this without needing original buff len using inflate

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

Distrotech Application Library . . . 27
Referenced Lockable Objects . . . 35
Hashed bucket linked lists of referenced objects . . . 45
Posix thread interface . . . 55
Network socket interface . . . 63
SSL socket support . . . 77
Unix domain sockets . . . 86
Multicast sockets . . . 90
Linux network interface functions . . . 94
INI Style config file Interface . . . 108
Radius client interface . . . 115
Micelaneous utilities. . . . 123
Hashing and digest functions . . . 132
MD5 Hashing and digest functions . . . 133
SHA1 Hashing and digest functions . . . 135
SHA2-256Hashing and digest functions . . . 137
SHA2-512 Hashing and digest functions . . . 139
IPv4 and IPv6 functions . . . 141
IPv4 functions . . . 145
IPv6 functions . . . 156
File utility functions . . . 160
Openldap/SASL Interface . . . 163
XML Interface . . . 184
XSLT Interface . . . 200
CURL Url interface. . . . 206
Zlib Interface . . . 216
Burtle Bob hash algorythim. . . . 219
IPv6 Nat Mapping . . . 234
Windows Support . . . 238
Distrotech Application Library (Todo) . . . 241
Linux Netfilter . . . 242
Connection Tracking . . . 243
Queue interface . . . 248

Chapter 10

Data Structure Index

10.1 Data Structures

Here are the data structures with brief descriptions:

basic_auth
Basic authentification structure . . . 253
blist_obj
Entry in a bucket list . . . 254
bucket_list
Bucket list, hold hashed objects in buckets . . . 255
bucket_loop
Bucket iterator . . . 256
config_category
Configuration file category . . . 258
config_entry
Configuration category entry . . . 258
config_file
Config file . . . 259
curl_post
HTTP post data structure . . . 260
curlbuf
Buffer containing the result of a curl transaction . . . 261
framework_core
Application framework data . . . 262
fwsocket
Socket data structure . . . 264
ifinfo
Data structure containing interface information . . . 266
ipaddr_req
IP Netlink IP addr request . . . 267
iplink_req
IP Netlink request . . . 268
ldap_add
LDAP Add structure . . . 269
ldap_attr
LDAP attirbute . . . 270
ldap_attrval
LDAP attribute value . . . 271
ldap_conn
LDAP connection . . . 272
ldap_entry
LDAP entry . . . 274

ldap_modify
LDAP Modify structure . . . 275
ldap_modreq
LDAP mod request . . . 276
ldap_modval
Linked list of mod values . . . 277
ldap_rdn
LDAP Relative distingushed name linked list . . . 278
ldap_results
LDAP results . . . 279
ldap_simple
LDAP Simple bind . . . 279
natmap
RFC6296 Nat map . . . 280
nfq_queue . . . 281
nfq_struct . . . 282
pseudohdr
IPv4 header structur to cast a packet too . . . 283
radius_connection
Radius connection . . . 284
radius_packet
Radius Packet . . . 285
radius_server
Radius Server . . . 287
radius_session
Radius session . . . 288
ref_obj
Internal structure of all referenced objects . . . 290
sasl_defaults
SASL Paramaters used in authentification . . . 292
socket_handler
Socket handling thread data . . . 293
sockstruct
Socket union describing all address types . . . 294
ssldata
SSL data structure for enabling encryption on sockets . . . 295
thread_pvt
Thread struct used to create threads data needs to be first element . . . 297
threadcontainer
Global threads data . . . 298
unixclient_sockthread
Unix socket client data structure . . . 299
unixserv_sockthread
Unix socket server data structure . . . 300
xml_attr
XML attribute name value pair . . . 302
xml_node
Reference to a XML Node . . . 302
xml_node_iter
Iterator to traverse nodes in a xpath . . . 304
xml_search
XML xpath search result . . . 304
xslt_doc
XSLT Document . . . 305
xslt_param
XSLT Parameter name/value pair . . . 306
zobj
Zlib buffer used for compression and decompression . . . 307

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

doxygen/examples/socket.c
: Echo server using 1 server and 2 clients . . . 314

src/config.c
: INI style config file interface . . . 325

src/curl.c
: CURL Interface . . . 330

src/fileutil.c
: File utilities to test files (fstat) . . . 336

src/interface.c
: Wrapper arround Linux libnetlink for managing network interfaces . . . 360

src/iputil.c
: IPv4 And IPv6 Utiliies . . . 370

src/libxml2.c
: XML Interface . . . 377

src/libxslt.c
: XSLT Interface . . . 386

src/lookup3.c
: By Bob Jenkins, May 2006, Public Domain . . . 389

src/main.c
: Application framework . . . 403

src/nf_ctrack.c
: Linux Netfilter Connection Tracking . . . 407

src/nf_queue.c
: Linux netfilter queue interface . . . 411

src/openldap.c
: Openldap/SASL Implementation . . . 417

src/radius.c
: Simple radius client implementation . . . 436

src/refobj.c
: Referenced Lockable Objects . . . 444

src/rfc6296.c
: Implementation of RFC6296 . . . 452

src/socket.c
: Allocate and initialise a socket for use as a client or server . . . 317

src/sslutil.c
: TLSv1 SSLv2 SSLv3 DTLSv1 support . . . 455

src/thread.c
: Functions for starting and managing threads . . . 465

src/unixsock.c
Attach a thread to a unix socket start a new thread on connect 471
src/util.c
Utilities commonly used 477
src/winiface.cpp
Various routines for supporting Windows also requires C++ 481
src/zlib.c
Simplified implementation of zlib functions 483
src/include/dtsapp.h
DTS Application library API Include file 350

Chapter 12

Module Documentation

12.1 Distrotech Application Library

A Collection of helper functions and wrapped up interfaces to other libraries.

Modules

- Referenced Lockable Objects

  Utilities for managing referenced lockable objects.
- Posix thread interface

  Functions for starting and managing threads.
- Network socket interface

  Allocate and initialise a socket for use as a client or server.
- Linux network interface functions

  Implement various interface routines from libnetlink.
- INI Style config file Interface

  Reads a ini config file into grouped hashed buckets.
- Radius client interface

  Simple implementation of experimental radius client.
- Micelaneous utilities.

  Utilities commonly used.
- IPv4 and IPv6 functions

  Helper functions for various calculations.
- File utility functions

  Convinece wrappers arround stat.
- Openldap/SASL Interface

  Functions to interface with a LDAP server.
- XML Interface

  Utilities for managing XML documents.
- CURL Url interface.

  Interface to libCURL.
- Zlib Interface

  Simplified implementation of zlib functions.
- Burtle Bob hash algorythim.

  lookup3.c, by Bob Jenkins, May 2006, Public Domain (Original Documentation)
- IPv6 Nat Mapping

  Implementation of RFC6296.
- Windows Support

  Support for building with mingw32 (Requires XP SP1+)

Files

- file dtsapp.h

  DTS Application library API Include file.
- file main.c

  Application framework.

Macros

- #define FRAMEWORK_MAIN(progname, name, email, www, year, runfile, flags, sighfunc)

  A macro to replace main() with initilization and daemonization code.
- #define ALLOC_CONST(const_var, val)

  Macro to assign values to char const.

Typedefs

- typedef int(∗ frameworkfunc)(int, char ∗∗)

  Framework callback function.
- typedef void(∗ syssighandler)(int, siginfo_t ∗, void ∗)

  Callback to user supplied signal handler.

Enumerations

- enum framework_flags { FRAMEWORK_FLAG_DAEMON = 1 << 0, FRAMEWORK_FLAG_NOGNU = 1 << 1, FRAMEWORK_FLAG_DAEMONLOCK = 1 << 2 }

  Application control flags.

Functions

- void printgnu (const char ∗pname, int year, const char ∗dev, const char ∗email, const char ∗www)

  Print a brief GNU copyright notice on console.
- void daemonize ()

  Daemonise the application using fork/exit.
- int lockpidfile (const char ∗runfile)

  Lock the run file in the framework application info.
- void framework_mkcore (char ∗progname, char ∗name, char ∗email, char ∗web, int year, char ∗runfile, int flags, syssighandler sigfunc)

  Initilise application data structure and return a reference.
- int framework_init (int argc, char ∗argv[], frameworkfunc callback)

  Initilise the application daemonise and join the manager thread.

12.1.1 Detailed Description

A Collection of helper functions and wrapped up interfaces to other libraries.

12.1.2 Macro Definition Documentation

12.1.2.1 #define ALLOC_CONST(*const_var, val*)

Value:

```
{ \
        char *tmp_char; \
        if (val) { \
            tmp_char = (char*)malloc(strlen(val) + 1); \
            strcpy(tmp_char, val); \
            const_var = (const char*)tmp_char; \
        } else { \
            const_var = NULL; \
        } \
    }
```

Macro to assign values to char const.

Definition at line 959 of file dtsapp.h.

Referenced by add_radserver(), framework_mkcore(), ldap_addinit(), ldap_modifyinit(), ldap_saslbind(), xml_modify(), and xslt_addparam().

12.1.2.2 #define FRAMEWORK_MAIN(*progname, name, email, www, year, runfile, flags, sighfunc*)

Value:

```
static int  framework_main(int argc, char *argv[]); \
int  main(int argc, char *argv[]) { \
    framework_mkcore(progname, name, email, www, year, runfile,
      flags, sighfunc); \
    return (framework_init(argc, argv, framework_main)); \
} \
static int  framework_main(int argc, char *argv[])
```

A macro to replace main() with initilization and daemonization code.

Note

Argument count is argc and arguments is array argv.

See Also

framework_flags
framework_mkcore()
framework_init()

Parameters

progname	Descriptive program name.
name	Copyright holders name.
email	Copyright holders email.
www	Web address.
year	Copyright year.
runfile	Application runfile.
flags	Application flags.

sighfunc	Signal handler function.

Definition at line 949 of file dtsapp.h.

12.1.3 Typedef Documentation

12.1.3.1 typedef int(∗ frameworkfunc)(int, char ∗∗)

Framework callback function.

Parameters

argc	Argument count.
argv	Argument array.

Returns

Application exit code.

Definition at line 219 of file dtsapp.h.

12.1.3.2 typedef void(∗ syssighandler)(int, siginfo_t ∗, void ∗)

Callback to user supplied signal handler.

Parameters

sig	Signal been handled.
si	Sa sigaction.
unused	Unused cast to void from ucontext_t

Definition at line 228 of file dtsapp.h.

12.1.4 Enumeration Type Documentation

12.1.4.1 enum framework_flags

Application control flags.

Enumerator

FRAMEWORK_FLAG_DAEMON Allow application daemonization.

FRAMEWORK_FLAG_NOGNU Dont print GNU copyright.

FRAMEWORK_FLAG_DAEMONLOCK Create lockfile on daemonize latter. Its possible you want to call daemonize latter and want the lockfile created then

Note

not compatible with FRAMEWORK_FLAG_DAEMON and has no effect FRAMEWORK_FLAG_DAEMON is set.

Definition at line 310 of file dtsapp.h.

```
00310                       {
00312     FRAMEWORK_FLAG_DAEMON       = 1 << 0,
00314     FRAMEWORK_FLAG_NOGNU         = 1 << 1,
00319     FRAMEWORK_FLAG_DAEMONLOCK   = 1 << 2
00320 };
```

12.1.5 Function Documentation

12.1.5.1 void daemonize ( )

Daemonise the application using fork/exit.

This should be run early before file descriptors and threads are started

See Also

FRAMEWORK_MAIN()

Warning

on failure the program will exit.

Todo WIN32 options is there a alternative for this.

Definition at line 94 of file main.c.

References framework_core::flags, framework_core::flock, FRAMEWORK_FLAG_DAEMONLOCK, lockpidfile(), objunref(), and framework_core::runfile.

Referenced by framework_init(), and FRAMEWORK_MAIN().

```
00094                           {
00095     struct framework_core *ci =  framework_core_info;
00096
00097 #ifndef __WIN32__
00098     pid_t   forkpid;
00099
00100     /* fork and die daemonize*/
00101     forkpid = fork();
00102     if (forkpid > 0) {
00103         /* im all grown up and can pass onto child*/
00104         exit(0);
00105     } else if (forkpid < 0) {
00106         /* could not fork*/
00107         exit(-1);
00108     }
00109
00110     setsid();
00111
00112     /* Dont want these as a daemon*/
00113     signal(SIGTSTP, SIG_IGN);
00114     signal(SIGCHLD, SIG_IGN);
00115 #endif
00116
00117     /*delayed lock file from FRAMEWORK_MAIN / framework_init*/
00118     if (ci && (ci->flags & FRAMEWORK_FLAG_DAEMONLOCK)) {
00119         if ((ci->flock = lockpidfile(ci->runfile)) < 0) {
00120             printf("Could not lock pid file Exiting\n");
00121             while(framework_core_info) {
00122                 objunref(framework_core_info);
00123             }
00124             exit (-1);
00125         }
00126         objunref(ci);
00127     }
00128 }
```

12.1.5.2 int framework_init (int *argc*, char * *argv[]*, frameworkfunc *callback*)

Initilise the application daemonise and join the manager thread.

Warning

failure to pass a callback will require running stopthreads and jointhreads.
framework information configured by framework_mkcore will be freed on exit.

Parameters

argc	Argument count argv[0] will be program name.
argv	Argument array.
callback	Function to pass control too.

Definition at line 260 of file main.c.

References daemonize(), framework_core::developer, framework_core::email, framework_core::flags, framework_core::flock, FRAMEWORK_FLAG_DAEMON, FRAMEWORK_FLAG_DAEMONLOCK, FRAMEWORK_FLAG_NOGNU, lockpidfile(), objref(), objunref(), printgnu(), framework_core::progname, framework_core::runfile, framework_core::sa, seedrand(), sslstartup(), stopthreads(), unrefconfigfiles(), framework_core::www, and framework_core::year.

```
00260                                                                    {
00261     struct framework_core *ci =  framework_core_info;
00262     int ret = 0;
00263 
00264     seedrand();
00265     sslstartup();
00266 
00267     /*prinit out a GNU licence summary*/
00268     if (ci && !(ci->flags & FRAMEWORK_FLAG_NOGNU)) {
00269         printgnu(ci->progname, ci->year, ci->developer, ci->
      email, ci->www);
00270     }
00271 
00272     /* grab a ref for framework_core_info to be used latter*/
00273     if (ci && ci->flags & FRAMEWORK_FLAG_DAEMONLOCK) {
00274         objref(ci);
00275     }
00276 
00277     /* fork the process to daemonize it*/
00278     if (ci && ci->flags & FRAMEWORK_FLAG_DAEMON) {
00279         daemonize();
00280     }
00281 
00282     /* write pid to lockfile this should be done post daemonize*/
00283     if (ci && !(ci->flags & FRAMEWORK_FLAG_DAEMONLOCK)) {
00284         if ((ci->flock = lockpidfile(ci->runfile)) < 0) {
00285             printf("Could not lock pid file Exiting\n");
00286             return -1;
00287         }
00288     }
00289 
00290 #ifndef __WIN32__
00291     /* interupt handler close clean on term so physical is reset*/
00292     configure_sigact(framework_core_info->sa);
00293 #endif
00294 
00295     /*run the code from the application*/
00296     if (callback) {
00297         ret = callback(argc, argv);
00298         /* wait for all threads to end*/
00299         stopthreads(1);
00300     }
00301 
00302     /* turn off the lights*/
00303     objunref(ci);
00304     if (framework_core_info && framework_core_info->flags &
      FRAMEWORK_FLAG_DAEMONLOCK) {
00305         objunref(framework_core_info);
00306     }
00307     unrefconfigfiles();
00308     return (ret);
00309 }
```

12.1.5.3 void framework_mkcore (char * *progname*, char * *name*, char * *email*, char * *web*, int *year*, char * *runfile*, int *flags*, syssighandler *sigfunc*)

Initilise application data structure and return a reference.

Warning

failure to supply a signal handler on non WIN32 systems will deafault to exiting with -1 on SIGINT/SIGKILL.

Todo does threads actually work in windows with no sighandler.

Warning

do not call this function without calling framework_init as the memory allocated will not be freed.

Parameters

progname	Descrioptive program name.
name	Copyright holder.
email	Copyright email address.
web	Website address.
year	Copyright year.
runfile	Run file that will store the pid and be locked (flock).
flags	Application flags.
sigfunc	Signal handler.

Definition at line 221 of file main.c.

References ALLOC_CONST, framework_core::developer, framework_core::email, framework_core::flags, objalloc(), objunref(), framework_core::progname, framework_core::runfile, framework_core::sa, framework_core::sig_handler, framework_core::www, and framework_core::year.

```
00221
                                    {
00222     struct framework_core *core_info;
00223     if (framework_core_info) {
00224         objunref(framework_core_info);
00225         framework_core_info = NULL;
00226     }
00227
00228     if (!(core_info = objalloc(sizeof(*core_info), framework_free))) {
00229         return;
00230     }
00231
00232 #ifndef __WIN32__
00233     if (core_info && !(core_info->sa = malloc(sizeof(*core_info->sa)))) {
00234         free(core_info);
00235         return;
00236     }
00237 #endif
00238
00239     ALLOC_CONST(core_info->developer, name);
00240     ALLOC_CONST(core_info->email, email);
00241     ALLOC_CONST(core_info->www, web);
00242     ALLOC_CONST(core_info->runfile, runfile);
00243     ALLOC_CONST(core_info->progname, progname);
00244     core_info->year = year;
00245     core_info->flags = flags;
00246 #ifndef __WIN32__
00247     core_info->sig_handler = sigfunc;
00248 #endif
00249     /* Pass reference to static system variable*/
00250     framework_core_info = core_info;
00251 }
```

12.1.5.4 int lockpidfile (const char * *runfile*)

Lock the run file in the framework application info.

This can be delayed till running daemonize in the user function loop setting flag FRAMEWORK_FLAG_DAEMONLOCK

Parameters

runfile	File to write pid to and lock.

Returns

0 if no file is specified or not supported. The file descriptor on success.

Definition at line 135 of file main.c.

References framework_core::flock.

Referenced by daemonize(), and framework_init().

```
00135                                                {
00136     int lck_fd = 0;
00137 #ifndef __WIN32__
00138     char pidstr[12];
00139     pid_t   mypid;
00140
00141     mypid = getpid();
00142     sprintf(pidstr,"%i\n", (int)mypid);
00143     if (runfile && ((lck_fd = open(runfile, O_RDWR|O_CREAT, 0640)) > 0) && (!
      flock(lck_fd, LOCK_EX | LOCK_NB))) {
00144         if (write(lck_fd, pidstr, strlen(pidstr)) < 0) {
00145             close(lck_fd);
00146             lck_fd = -1;
00147         }
00148     /* file was opened and not locked*/
00149     } else if (runfile && lck_fd) {
00150         close(lck_fd);
00151         lck_fd = -1;
00152     }
00153 #endif
00154     return (lck_fd);
00155 }
```

12.1.5.5 void printgnu (const char * *pname*, int *year*, const char * *dev*, const char * *email*, const char * *www*)

Print a brief GNU copyright notice on console.

See Also

FRAMEWORK_MAIN()
framework_mkcore()

Parameters

pname	Detailed application name.
year	Copyright year.
dev	Programer / copyright holder name.
email	Email address.
www	HTTP URL.

Definition at line 78 of file main.c.

Referenced by framework_init().

```
00078                                                                                {
00079     printf("\n"
00080            "    %s\n\n"
00081            "    Copyright (C) %i %s <%s>\n\n"
00082            "        %s\n\n"
00083            "    This program comes with ABSOLUTELY NO WARRANTY\n"
00084            "    This is free software, and you are welcome to redistribute it\n"
00085            "    under certain conditions.\n\n", pname, year, dev, email,
      www);
00086 }
```

12.2 Referenced Lockable Objects

Utilities for managing referenced lockable objects.

Modules

- Hashed bucket linked lists of referenced objects

  Store references in and retrive from linked lists based on a hash.

Files

- file refobj.c

  Referenced Lockable Objects.

Data Structures

- struct ref_obj

  Internal structure of all referenced objects.

Macros

- #define clearflag(obj, flag)

  Atomically clear a flag in the flags field of a referenced object.
- #define setflag(obj, flag)

  Atomically set a flag in the flags field of a referenced object.
- #define testflag(obj, flag) (objlock(obj) | (obj->flags & flag) | objunlock(obj))

  Atomically test a flag in the flags field of a referenced object.
- #define DTS_OJBREF_CLASS(classtype)

  Add this macro to a C++ class to add refobj support.
- #define REFOBJ_MAGIC 0xdeadc0de

  Magic number stored as first field of all referenced objects.
- #define refobj_offset sizeof(struct ref_obj);

  The size of ref_obj is the offset for the data.

Typedefs

- typedef void(* objdestroy)(void *)

  Callback used to clean data of a reference object when it is to be freed.

Functions

- void * objalloc (int size, objdestroy destructor)

  Allocate a referenced lockable object.
- int objref (void *data)

  Reference a object.
- int objunref (void *data)

  Drop reference held.
- int objcnt (void *data)

  Return current reference count.

- int objsize (void *data)

  Size requested for data.
- int objlock (void *data)

  Lock the reference.
- int objtrylock (void *data)

  Try lock a reference.
- int objunlock (void *data)

  Unlock a reference.
- void * objchar (const char *orig)

  Return a reference to copy of a buffer.

12.2.1 Detailed Description

Utilities for managing referenced lockable objects.

See Also

Referenced Lockable Objects

12.2.2 Macro Definition Documentation

12.2.2.1 #define clearflag(*obj*, *flag*)

Value:

```
objlock(obj);\
obj->flags &= ~flag;\
objunlock(obj)
```

Atomically clear a flag in the flags field of a referenced object.

Definition at line 918 of file dtsapp.h.

12.2.2.2 #define DTS_OJBREF_CLASS(*classtype*)

Value:

```
void *operator new(size_t sz) {\
    return objalloc(sz, &classtype::dts_unref_classtype);\
}\
void operator delete(void *obj) {\
}\
static void dts_unref_classtype(void *data) {\
    delete (classtype*)data;\
}\
~classtype()
```

Add this macro to a C++ class to add refobj support.

This macro defines operator overloads for new/delete and declares a destructor.

Note

this should not be used with inheritance

Definition at line 976 of file dtsapp.h.

12.2.2.3 #define REFOBJ_MAGIC 0xdeadc0de

Magic number stored as first field of all referenced objects.

Definition at line 34 of file refobj.c.

Referenced by objalloc(), objcnt(), objlock(), objref(), objsize(), objtrylock(), objunlock(), and objunref().

12.2.2.4 #define refobj_offset sizeof(struct ref_obj);

The size of ref_obj is the offset for the data.

Definition at line 119 of file refobj.c.

Referenced by addtobucket(), objalloc(), objcnt(), objlock(), objref(), objsize(), objtrylock(), objunlock(), and objunref().

12.2.2.5 #define setflag(*obj, flag*)

Value:

```
objlock(obj);\
obj->flags |= flag; \
objunlock(obj)
```

Atomically set a flag in the flags field of a referenced object.

Definition at line 925 of file dtsapp.h.

Referenced by close_socket(), dtls_listenssl(), framework_mkthread(), jointhreads(), nf_ctrack_endtrace(), socketwrite_d(), stopthreads(), and tlsaccept().

12.2.2.6 #define testflag(*obj, flag*) (objlock(obj) | (obj->flags & flag) | objunlock(obj))

Atomically test a flag in the flags field of a referenced object.

Definition at line 932 of file dtsapp.h.

Referenced by framework_mkthread(), framework_threadok(), socketread_d(), and socketwrite_d().

12.2.3 Typedef Documentation

12.2.3.1 typedef void(* objdestroy)(void *)

Callback used to clean data of a reference object when it is to be freed.

Parameters

data	Data held by reference about to be freed.

Definition at line 264 of file dtsapp.h.

12.2.4 Function Documentation

12.2.4.1 void* objalloc (int *size,* objdestroy *destructor*)

Allocate a referenced lockable object.

Use malloc to allocate memory to contain the data lock and reference the lock is initialised magic and reference set. The data begins at the end of the ref_obj set a pointer to it and return.

Parameters

size	Size of the data buffer to allocate in addition to the reference.
destructor	Function called before the memory is freed to cleanup.

Returns

Pointer to a data buffer size big.

Definition at line 129 of file refobj.c.

References ref_obj::cnt, ref_obj::data, ref_obj::destroy, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, refobj_offset, and ref_obj::size.

Referenced by accept_socket(), add_radserver(), b64enc_buf(), create_bucketlist(), create_kernmac(), create_kernvlan(), curl_newauth(), curl_newpost(), curl_setauth_cb(), curl_setprogress(), curlinit(), dtls_listenssl(), framework_mkcore(), framework_mkthread(), get_ifinfo(), init_bucket_loop(), ldap_addinit(), ldap_connect(), ldap_modifyinit(), ldap_saslbind(), ldap_simplebind(), make_socket(), nfqueue_attach(), objchar(), rfc6296_map_add(), set_interface_addr(), set_interface_flags(), set_interface_ipaddr(), set_interface_name(), startthreads(), tlsaccept(), unixsocket_server(), xml_doctobuffer(), xml_getfirstnode(), xml_init(), xml_loadbuf(), xml_loaddoc(), xml_xpath(), xslt_addparam(), xslt_apply_buffer(), xslt_init(), xslt_open(), and zcompress().

```
00129                                                   {
00130     struct ref_obj *ref;
00131     int asize;
00132     char *robj;
00133
00134     asize  = size + refobj_offset;
00135
00136     if ((robj = malloc(asize))) {
00137         memset(robj, 0, asize);
00138         ref = (struct ref_obj *)robj;
00139         pthread_mutex_init(&ref->lock, NULL);
00140         ref->magic = REFOBJ_MAGIC;
00141         ref->cnt = 1;
00142         ref->data = robj + refobj_offset;
00143         ref->size = asize;
00144         ref->destroy = destructor;
00145         return (ref->data);
00146     }
00147     return NULL;
00148 }
```

12.2.4.2 void* objchar (const char * *orig*)

Return a reference to copy of a buffer.

Parameters

orig	Original buffer to copy.

Returns

Reference to new instance of orig.

Definition at line 330 of file refobj.c.

References objalloc().

```
00330                                       {
00331     int len = strlen(orig) + 1;
00332     void *nobj;
00333
00334     if ((nobj = objalloc(len, NULL))) {
00335         memcpy(nobj, orig, len);
00336     }
00337     return nobj;
00338 }
```

12.2.4.3 int objcnt (void * *data*)

Return current reference count.

Parameters

data	Pointer to determine active reference count.

Returns

-1 on error or the current count.

Definition at line 222 of file refobj.c.

References ref_obj::cnt, ref_obj::data, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, and refobj_offset.

Referenced by ldap_unref_attr(), and ldap_unref_entry().

```
00222                                {
00223     char *ptr = data;
00224     int ret = -1;
00225     struct ref_obj *ref;
00226
00227     if (!data) {
00228         return (ret);
00229     }
00230
00231     ptr = ptr - refobj_offset;
00232     ref = (struct ref_obj *)ptr;
00233
00234     if (ref->magic == REFOBJ_MAGIC) {
00235         pthread_mutex_lock(&ref->lock);
00236         ret = ref->cnt;
00237         pthread_mutex_unlock(&ref->lock);
00238     }
00239     return (ret);
00240 }
```

12.2.4.4 int objlock (void * *data*)

Lock the reference.

Parameters

data	Reference to lock

Returns

Always returns 0 will only lock if a valid object.

Definition at line 269 of file refobj.c.

References ref_obj::data, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, and refobj_offset.

Referenced by accept_socket(), addtobucket(), bucket_list_cnt(), create_kernmac(), create_kernvlan(), curl_postitem(), curlinit(), dtls_listenssl(), dtlshandltimeout(), dtlstimeout(), dtsl_serveropts(), framework_mkthread(), get_iface_index(), jointhreads(), ldap_doadd(), ldap_domodify(), ldap_saslbind(), ldap_simplebind(), nf_ctrack_delete(), nf_ctrack_dump(), nf_ctrack_nat(), nfqueue_attach(), remove_bucket_item(), remove_bucket_loop(), set_interface_addr(), set_interface_flags(), set_interface_ipaddr(), set_interface_name(), socketread_d(), socketserver(), socketwrite_d(), ssl_shutdown(), stopthreads(), url_escape(), url_unescape(), xml_addnode(), xml_appendnode(), xml_createpath(), xml_delete(), xml_doctobuffer(), xml_getfirstnode(), xml_getnextnode(), xml_getrootnode(), xml_modify(), xml_savefile(), xml_setattr(), xml_unlink(), xml_xpath(), xslt_addparam(), xslt_apply(), xslt_apply_buffer(), and xslt_clearparam().

```
00269                                 {
00270     char *ptr = data;
00271     struct ref_obj *ref;
00272
00273     ptr = ptr - refobj_offset;
00274     ref = (struct ref_obj *)ptr;
00275
00276     if (data && ref->magic == REFOBJ_MAGIC) {
00277         pthread_mutex_lock(&ref->lock);
00278     }
00279     return (0);
00280 }
```

12.2.4.5 int objref (void * *data*)

Reference a object.

Parameters

data	Data to obtain reference for.

Returns

0 on error or the current count (after incrementing)

Definition at line 153 of file refobj.c.

References ref_obj::cnt, ref_obj::data, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, and refobj_offset.

Referenced by addtobucket(), bucket_list_find_key(), create_kernmac(), create_kernvlan(), curl_setauth_cb(), curl_setprogress(), curlinit(), framework_init(), framework_mkthread(), get_category_next(), get_config_category(), get_config_file(), get_iface_index(), ifhwaddr(), init_bucket_loop(), jointhreads(), ldap_domodify(), ldap_saslbind(), ldap_simplebind(), ldap_simplerebind(), mcast_socket(), next_bucket_loop(), set_interface_addr(), set_interface_flags(), set_interface_ipaddr(), set_interface_name(), startthreads(), stopthreads(), unixsocket_client(), unixsocket_server(), xml_addnode(), xml_appendnode(), xml_createpath(), xml_getfirstnode(), xml_getnextnode(), xml_getnodes(), xml_init(), xml_xpath(), xslt_addparam(), and xslt_init().

```
00153                                {
00154     char *ptr = data;
00155     struct ref_obj *ref;
00156     int ret = 0;
00157
00158     ptr = ptr - refobj_offset;
00159     ref = (struct ref_obj *)ptr;
00160
00161     if (!data || !ref || (ref->magic != REFOBJ_MAGIC)) {
00162         return (ret);
00163     }
00164
00165     /*double check just incase im gone*/
00166     if (!pthread_mutex_lock(&ref->lock)) {
00167         if ((ref->magic == REFOBJ_MAGIC) && (ref->cnt > 0)) {
00168             ref->cnt++;
00169             ret = ref->cnt;
00170         }
00171         pthread_mutex_unlock(&ref->lock);
00172     }
00173
00174     return (ret);
00175 }
```

12.2.4.6 int objsize (void * *data*)

Size requested for data.

Note

the size of the data is returned.

Parameters

data	Pointer to data to obtain size of.

Returns

size requested for allocation not allocation [excludes refobj].

Definition at line 246 of file refobj.c.

References ref_obj::data, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, refobj_offset, and ref_obj::size.

```
00246                                {
00247     char *ptr = data;
00248     int ret = 0;
00249     struct ref_obj *ref;
00250
00251     if (!data) {
00252         return (ret);
00253     }
00254
00255     ptr = ptr - refobj_offset;
00256     ref = (struct ref_obj *)ptr;
00257
00258     if (ref->magic == REFOBJ_MAGIC) {
00259         pthread_mutex_lock(&ref->lock);
00260         ret = ref->size - refobj_offset;
00261         pthread_mutex_unlock(&ref->lock);
00262     }
00263     return (ret);
00264 }
```

12.2.4.7 int objtrylock (void * *data*)

Try lock a reference.

Parameters

data	Reference to attempt to lock.

Returns

0 on success -1 on failure.

Definition at line 285 of file refobj.c.

References ref_obj::data, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, and refobj_offset.

```
00285                                   {
00286     char *ptr = data;
00287     struct ref_obj *ref;
00288
00289     ptr = ptr - refobj_offset;
00290     ref = (struct ref_obj *)ptr;
00291
00292     if (ref->magic == REFOBJ_MAGIC) {
00293         return ((pthread_mutex_trylock(&ref->lock)) ? -1 : 0);
00294     }
00295     return (-1);
00296 }
```

12.2.4.8 int objunlock (void * *data*)

Unlock a reference.

Parameters

data	Reference to unlock.

Returns

Always returns 0.

Definition at line 301 of file refobj.c.

References ref_obj::data, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, and refobj_offset.

Referenced by accept_socket(), addtobucket(), bucket_list_cnt(), create_kernmac(), create_kernvlan(), curl_postitem(), curlinit(), dtls_listenssl(), dtlshandltimeout(), dtlstimeout(), dtsl_serveropts(), framework_mkthread(), get_iface_index(), jointhreads(), ldap_doadd(), ldap_domodify(), ldap_saslbind(), ldap_simplebind(), nf_ctrack_delete(), nf_ctrack_dump(), nf_ctrack_nat(), nfqueue_attach(), remove_bucket_item(), remove_bucket_loop(),

set_interface_addr(), set_interface_flags(), set_interface_ipaddr(), set_interface_name(), socketread_d(), socketserver(), socketwrite_d(), ssl_shutdown(), stopthreads(), url_escape(), url_unescape(), xml_addnode(), xml_appendnode(), xml_createpath(), xml_delete(), xml_doctobuffer(), xml_getfirstnode(), xml_getnextnode(), xml_getrootnode(), xml_modify(), xml_savefile(), xml_setattr(), xml_unlink(), xml_xpath(), xslt_addparam(), xslt_apply(), xslt_apply_buffer(), and xslt_clearparam().

```
00301                                   {
00302     char *ptr = data;
00303     struct ref_obj *ref;
00304
00305     ptr = ptr - refobj_offset;
00306     ref = (struct ref_obj *)ptr;
00307
00308     if (ref->magic == REFOBJ_MAGIC) {
00309         pthread_mutex_unlock(&ref->lock);
00310     }
00311     return (0);
00312 }
```

12.2.4.9 int objunref (void * *data*)

Drop reference held.

If the reference is the last reference call the destructor to clean up and then free the memory used.

Warning

The reference should not be used again and ideally set to NULL.

Parameters

data	Data we are droping a reference for

Returns

-1 on error or the refrence count after decrementing.

Definition at line 184 of file refobj.c.

References ref_obj::cnt, ref_obj::data, ref_obj::destroy, ref_obj::lock, ref_obj::magic, REFOBJ_MAGIC, refobj_offset, and ref_obj::size.

Referenced by accept_socket(), add_radserver(), addtobucket(), bucket_list_find_key(), bucketlist_callback(), close_socket(), closenetlink(), create_kernmac(), create_kernvlan(), curl_setauth_cb(), curl_setprogress(), curlclose(), curlinit(), daemonize(), dtls_listenssl(), framework_init(), framework_mkcore(), framework_mkthread(), framework_threadok(), get_category_loop(), get_category_next(), get_config_category(), get_config_file(), get_iface_index(), ifhwaddr(), jointhreads(), ldap_add_attr(), ldap_addinit(), ldap_connect(), ldap_doadd(), ldap_domodify(), ldap_mod_add(), ldap_mod_addattr(), ldap_mod_del(), ldap_mod_delattr(), ldap_mod_rep(), ldap_mod_repattr(), ldap_modifyinit(), ldap_saslbind(), ldap_simplebind(), ldap_simplerebind(), ldap_unref_attr(), ldap_unref_entry(), make_socket(), mcast_socket(), nf_ctrack_close(), nf_ctrack_endtrace(), nf_ctrack_trace(), nfqueue_attach(), process_config(), remove_bucket_item(), remove_bucket_loop(), rfc6296_map_add(), rfc6296_test(), set_interface_addr(), set_interface_flags(), set_interface_ipaddr(), set_interface_name(), socketread_d(), socketwrite_d(), startthreads(), stopthreads(), thread_signal(), unixsocket_client(), unixsocket_server(), unrefconfigfiles(), url_escape(), url_unescape(), xml_addnode(), xml_appendnode(), xml_close(), xml_createpath(), xml_getattr(), xml_getfirstnode(), xml_getnextnode(), xml_loadbuf(), xml_loaddoc(), xml_xpath(), xslt_addparam(), xslt_apply(), xslt_apply_buffer(), xslt_clearparam(), and xslt_close().

```
00184                                   {
00185     char *ptr = data;
00186     struct ref_obj *ref;
00187     int ret = -1;
00188
00189     if (!data) {
00190         return (ret);
00191     }
```

```
00192
00193     ptr = ptr - refobj_offset;
00194     ref = (struct ref_obj *)ptr;
00195
00196     if ((ref->magic == REFOBJ_MAGIC) && (ref->cnt)) {
00197         pthread_mutex_lock(&ref->lock);
00198         ref->cnt--;
00199         ret = ref->cnt;
00200         /* free the object its no longer in use*/
00201         if (!ret) {
00202             ref->magic = 0;
00203             ref->size = 0;
00204             ref->data = NULL;
00205             if (ref->destroy) {
00206                 ref->destroy(data);
00207             }
00208             pthread_mutex_unlock(&ref->lock);
00209             pthread_mutex_destroy(&ref->lock);
00210             free(ref);
00211         } else {
00212             pthread_mutex_unlock(&ref->lock);
00213         }
00214     }
00215     return (ret);
00216 }
```

12.3 Hashed bucket linked lists of referenced objects

Store references in and retrive from linked lists based on a hash.

Files

- file refobj.c

  Referenced Lockable Objects.

Data Structures

- struct blist_obj

  Entry in a bucket list.
- struct bucket_list

  Bucket list, hold hashed objects in buckets.
- struct bucket_loop

  Bucket iterator.

Typedefs

- typedef int32_t(∗ blisthash)(const void ∗, int)

  Callback used to calculate the hash of a structure.
- typedef void(∗ blist_cb)(void ∗, void ∗)

  This callback is run on each entry in a list.

Functions

- void ∗ create_bucketlist (int bitmask, blisthash hash_function)
- int addtobucket (struct bucket_list ∗blist, void ∗data)

  Add a reference to the bucketlist.
- void remove_bucket_item (struct bucket_list ∗blist, void ∗data)

  Remove and unreference a item from the list.
- int bucket_list_cnt (struct bucket_list ∗blist)

  Return number of items in the list.
- void ∗ bucket_list_find_key (struct bucket_list ∗blist, const void ∗key)

  Find and return a reference to a item matching supplied key.
- void bucketlist_callback (struct bucket_list ∗blist, blist_cb callback, void ∗data2)

  Run a callback function on all items in the list.
- struct bucket_loop ∗ init_bucket_loop (struct bucket_list ∗blist)

  Create a bucket list iterator to safely iterate the list.
- void ∗ next_bucket_loop (struct bucket_loop ∗bloop)

  Return a reference to the next item in the list this could be the first item.
- void remove_bucket_loop (struct bucket_loop ∗bloop)

  Safely remove a item from a list while iterating in a loop.

12.3.1 Detailed Description

Store references in and retrive from linked lists based on a hash. Create a hashed bucket list.

See Also

Hashed Bucket Lists
Burtle Bob hash algorythim.

A bucket list is a ref obj the "list" element is a array of "bucket" entries each has a hash the default is to hash the memory when there is no call back

Todo Dont hash the memory supply a key perhaps a key array type.

Warning

the hash must be calculated on immutable data.

Note

a bucket list should only contain objects of the same type.
Unreferencing the bucketlist will cause it to be emptied and freed when the count reaches 0.

See Also

blisthash

Parameters

bitmask	Number of buckets to create 2^bitmask.
hash_function	Callback that returns the unique hash for a item this value must not change.

Returns

Reference to a empty bucket list.

12.3.2 Typedef Documentation

12.3.2.1 typedef void(* blist_cb)(void *, void *)

This callback is run on each entry in a list.

See Also

bucketlist_callback()

Parameters

data	Reference held by the list.
data2	Reference to data supplied when calling bucketlist_callback.

Definition at line 278 of file dtsapp.h.

12.3.2.2 typedef int32_t(* blisthash)(const void *, int)

Callback used to calculate the hash of a structure.

Parameters

data	Data or key to calculate hash from.
key	Key if set to non zero data supplied is the key not data.

Returns

Hash for the Reference.

Definition at line 271 of file dtsapp.h.

12.3.3 Function Documentation

12.3.3.1 int addtobucket (struct bucket_list * *blist*, void * *data*)

Add a reference to the bucketlist.

Create a entry in the list for reference obtained from data.

Parameters

blist	Bucket list to add too.
data	to obtain a reference too and add to the list.

Returns

0 on failure 1 on success.

Definition at line 428 of file refobj.c.

References bucket_list::bucketbits, bucket_list::count, ref_obj::data, blist_obj::data, blist_obj::hash, bucket_list::list, bucket_list::locks, blist_obj::next, objlock(), objref(), objunlock(), objunref(), blist_obj::prev, refobj_offset, and bucket_list::version.

Referenced by add_radserver(), framework_mkthread(), process_config(), rfc6296_map_add(), and xslt_addparam().

```
00428                                                                   {
00429     char *ptr = data;
00430     struct ref_obj *ref;
00431     struct blist_obj *lhead, *tmp;
00432     unsigned int hash, bucket;
00433
00434     if (!objref(blist)) {
00435         return (0);
00436     }
00437
00438     if (!objref(data)) {
00439         objunref(blist);
00440         return (0);
00441     }
00442
00443     ptr = ptr - refobj_offset;
00444     ref = (struct ref_obj *)ptr;
00445
00446     hash = gethash(blist, data, 0);
00447     bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
      bucketbits) - 1));
00448
00449     pthread_mutex_lock(&blist->locks[bucket]);
00450     lhead = blist->list[bucket];
00451     /*no head or non null head*/
00452     if (!lhead || lhead->prev) {
00453         if (!(tmp = malloc(sizeof(*tmp)))) {
00454             pthread_mutex_unlock(&blist->locks[bucket]);
00455             objunref(data);
00456             objunref(blist);
00457             return (0);
00458         }
00459         memset(tmp, 0, sizeof(*tmp));
00460         tmp->hash = hash;
```

```
00461         tmp->data = ref;
00462
00463         /*there is no head*/
00464         if (!lhead) {
00465             blist->list[bucket] = tmp;
00466             tmp->prev = tmp;
00467             tmp->next = NULL;
00468         /*become new head*/
00469         } else if (hash < lhead->hash) {
00470             tmp->next = lhead;
00471             tmp->prev = lhead->prev;
00472             lhead->prev = tmp;
00473             blist->list[bucket] = tmp;
00474         /*new tail*/
00475         } else if (hash > lhead->prev->hash) {
00476             tmp->prev = lhead->prev;
00477             tmp->next = NULL;
00478             lhead->prev->next = tmp;
00479             lhead->prev = tmp;
00480         /*insert entry*/
00481         } else {
00482             lhead = blist_gotohash(lhead, hash, blist->bucketbits);
00483             tmp->next = lhead->next;
00484             tmp->prev = lhead;
00485
00486             if (lhead->next) {
00487                 lhead->next->prev = tmp;
00488             } else {
00489                 blist->list[bucket]->prev = tmp;
00490             }
00491             lhead->next = tmp;
00492         }
00493     } else {
00494         /*set NULL head*/
00495         lhead->data = ref;
00496         lhead->prev = lhead;
00497         lhead->next = NULL;
00498         lhead->hash = hash;
00499     }
00500
00501     blist->version[bucket]++;
00502     pthread_mutex_unlock(&blist->locks[bucket]);
00503
00504     objlock(blist);
00505     blist->count++;
00506     objunlock(blist);
00507     objunref(blist);
00508
00509     return (1);
00510 }
```

12.3.3.2 int bucket_list_cnt (struct bucket_list * *blist*)

Return number of items in the list.

Parameters

blist	Bucket list to get count of.

Returns

Total number of items in all buckets.

Definition at line 552 of file refobj.c.

References bucket_list::count, objlock(), and objunlock().

Referenced by add_radserver(), ldap_doadd(), and ldap_domodify().

```
00552                                                   {
00553     int ret = -1;
00554
00555     if (blist) {
00556         objlock(blist);
00557         ret = blist->count;
00558         objunlock(blist);
00559     }
00560     return (ret);
00561 }
```

12.3.3.3 void* bucket_list_find_key (struct bucket_list * *blist,* const void * *key*)

Find and return a reference to a item matching supplied key.

The key is supplied to the hash callback ad the data value and the key flag set. The hash for the object will be returned by the hash callback to find the item in the lists.

Note

if the hash is not calculated equal to the original value it wont be found.

Parameters

blist	Bucket list to search.
key	Supplied to hash callback to find the item.

Returns

New reference to the found item that needs to be unreferenced or NULL.

Definition at line 572 of file refobj.c.

References bucket_list::bucketbits, ref_obj::data, blist_obj::data, blist_obj::hash, bucket_list::list, bucket_list::locks, objref(), and objunref().

Referenced by get_config_category(), get_config_entry(), get_config_file(), ldap_getattr(), ldap_getentry(), nfqueue_attach(), xml_getattr(), and xml_getnode().

```
00572                                                                        {
00573     struct blist_obj *entry;
00574     int hash, bucket;
00575
00576     if (!blist) {
00577         return (NULL);
00578     }
00579
00580     hash = gethash(blist, key, 1);
00581     bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
     bucketbits) - 1));
00582
00583     pthread_mutex_lock(&blist->locks[bucket]);
00584     entry = blist_gotohash(blist->list[bucket], hash + 1, blist->bucketbits);
00585     if (entry && entry->data) {
00586         objref(entry->data->data);
00587     } else
00588         if (!entry) {
00589             pthread_mutex_unlock(&blist->locks[bucket]);
00590             return NULL;
00591         }
00592
00593     pthread_mutex_unlock(&blist->locks[bucket]);
00594
00595     if (entry->data && (entry->hash == hash)) {
00596         return (entry->data->data);
00597     } else
00598         if (entry->data) {
00599             objunref(entry->data->data);
00600         }
00601
00602     return NULL;
00603 }
```

12.3.3.4 void bucketlist_callback (struct bucket_list * *blist,* blist_cb *callback,* void * *data2*)

Run a callback function on all items in the list.

This will iterate safely through all items calling the callback with the item and the optional data supplied.

See Also

blist_cb

Parameters

blist	Bucket list to iterate through.
callback	Callback to call for each iteration.
data2	Data to be set as option to the callback.

Definition at line 613 of file refobj.c.

References init_bucket_loop(), next_bucket_loop(), and objunref().

Referenced by config_cat_callback(), config_entry_callback(), config_file_callback(), and rfc6296_test().

```
00613                                                                                           {
00614     struct bucket_loop *bloop;
00615     void *data;
00616 
00617     if (!blist || !callback) {
00618         return;
00619     }
00620 
00621     bloop = init_bucket_loop(blist);
00622     while(blist && bloop && (data = next_bucket_loop(bloop))) {
00623         callback(data, data2);
00624         objunref(data);
00625     }
00626     objunref(bloop);
00627 }
```

12.3.3.5 void* create_bucketlist (int *bitmask,* blisthash *hash_function*)

Definition at line 356 of file refobj.c.

References bucket_list::bucketbits, and objalloc().

Referenced by add_radserver(), ldap_addinit(), ldap_modifyinit(), rfc6296_map_add(), socketserver(), startthreads(), xslt_clearparam(), and xslt_open().

```
00356                                                                            {
00357     struct bucket_list *new;
00358     short int buckets, cnt;
00359 
00360     buckets = (1 << bitmask);
00361 
00362     /* allocate session bucket list memory size of the struct plus a list lock and version for each bucket
      */
00363     if (!(new = objalloc(sizeof(*new) + (sizeof(void *) + sizeof(pthread_mutex_t) + sizeof(size_t))
       * buckets, empty_buckets))) {
00364         return NULL;
00365     }
00366 
00367     /*initialise each bucket*/
00368     new->bucketbits = bitmask;
00369     new->list = (void *)((char *)new + sizeof(*new));
00370     for (cnt = 0; cnt < buckets; cnt++) {
00371         if ((new->list[cnt] = malloc(sizeof(*new->list[cnt])))) {
00372             memset(new->list[cnt], 0, sizeof(*new->list[cnt]));
00373         }
00374     }
00375 
00376     /*next pointer is pointer to locks*/
00377     new->locks = (void *)&new->list[buckets];
00378     for (cnt = 0; cnt < buckets; cnt++) {
00379         pthread_mutex_init(&new->locks[cnt], NULL);
00380     }
00381 
00382     /*Next up version array*/
00383     new->version = (void *)&new->locks[buckets];
00384 
00385     new->hash_func = hash_function;
00386 
00387     return (new);
00388 }
```

12.3.3.6 struct bucket_loop* init_bucket_loop (struct bucket_list * *blist*)

Create a bucket list iterator to safely iterate the list.

Parameters

blist	Bucket list to create iterator for.

Returns

Bucket list iterator that needs to be unreferenced when completed.

Definition at line 640 of file refobj.c.

References bucket_loop::blist, bucket_loop::bucket, blist_obj::hash, bucket_loop::head, bucket_loop::head_hash, bucket_list::list, bucket_list::locks, objalloc(), objref(), bucket_list::version, and bucket_loop::version.

Referenced by bucketlist_callback(), get_category_loop(), ldap_doadd(), and ldap_domodify().

```
00640                                                                    {
00641     struct bucket_loop *bloop = NULL;
00642
00643     if (blist && (bloop = objalloc(sizeof(*bloop), free_bloop))) {
00644         objref(blist);
00645         bloop->blist = blist;
00646         bloop->bucket = 0;
00647         pthread_mutex_lock(&blist->locks[bloop->bucket]);
00648         bloop->head = blist->list[0];
00649         if (bloop->head) {
00650             bloop->head_hash = bloop->head->hash;
00651         };
00652         bloop->version = blist->version[0];
00653         pthread_mutex_unlock(&blist->locks[bloop->bucket]);
00654     }
00655
00656     return (bloop);
00657 }
```

12.3.3.7 void* next_bucket_loop (struct bucket_loop * *bloop*)

Return a reference to the next item in the list this could be the first item.

Parameters

bloop	Bucket iterator

Returns

Next available item or NULL when there no items left

Definition at line 662 of file refobj.c.

References bucket_loop::blist, bucket_loop::bucket, bucket_list::bucketbits, bucket_loop::cur, bucket_loop::cur_hash, ref_obj::data, blist_obj::data, blist_obj::hash, bucket_loop::head, bucket_loop::head_hash, bucket_list::list, bucket_list::locks, blist_obj::next, objref(), blist_obj::prev, bucket_list::version, and bucket_loop::version.

Referenced by bucketlist_callback(), get_category_next(), ldap_doadd(), and ldap_domodify().

```
00662                                                        {
00663     struct bucket_list *blist = bloop->blist;
00664     struct ref_obj *entry = NULL;
00665     void *data = NULL;
00666
00667     pthread_mutex_lock(&blist->locks[bloop->bucket]);
00668     if (bloop->head_hash && (blist->version[bloop->bucket] != bloop->
      version)) {
00669         /* bucket has changed unexpectedly i need to ff/rew to hash*/
00670         bloop->head = blist_gotohash(blist->list[bloop->bucket], bloop->
      head_hash + 1, blist->bucketbits);
00671         /*if head has gone find next suitable ignore any added*/
00672         while (bloop->head && (bloop->head->hash < bloop->head_hash)) {
00673             bloop->head = bloop->head->next;
00674         }
00675     }
00676
```

```
00677     while (!bloop->head || !bloop->head->prev) {
00678         pthread_mutex_unlock(&blist->locks[bloop->bucket]);
00679         bloop->bucket++;
00680         if (bloop->bucket < (1 << blist->bucketbits)) {
00681             pthread_mutex_lock(&blist->locks[bloop->bucket]);
00682             bloop->head = blist->list[bloop->bucket];
00683         } else {
00684             return NULL;
00685         }
00686     }
00687
00688     if (bloop->head) {
00689         bloop->cur = bloop->head;
00690         entry = (bloop->head->data) ? bloop->head->data : NULL;
00691         data = (entry) ? entry->data : NULL;
00692         objref(data);
00693         bloop->head = bloop->head->next;
00694         bloop->head_hash = (bloop->head) ? bloop->head->hash : 0;
00695         bloop->cur_hash = (bloop->cur) ? bloop->cur->hash : 0;
00696     }
00697     pthread_mutex_unlock(&blist->locks[bloop->bucket]);
00698
00699     return (data);
00700 }
```

12.3.3.8 void remove_bucket_item (struct bucket_list * *blist*, void * *data*)

Remove and unreference a item from the list.

Note

Dont use this function directly during iteration as it imposes performance penalties.

Parameters

blist	Bucket list to remove item from.

See Also

remove_bucket_loop

Parameters

data	Reference to be removed and unreferenced.

Definition at line 517 of file refobj.c.

References bucket_list::bucketbits, bucket_list::count, ref_obj::data, blist_obj::data, blist_obj::hash, bucket_list::list, bucket_list::locks, blist_obj::next, objlock(), objunlock(), objunref(), blist_obj::prev, and bucket_list::version.

Referenced by ldap_unref_attr(), and ldap_unref_entry().

```
00517                                                                   {
00518     struct blist_obj *entry;
00519     int hash, bucket;
00520
00521     hash = gethash(blist, data, 0);
00522     bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
      bucketbits) - 1));
00523
00524     pthread_mutex_lock(&blist->locks[bucket]);
00525     entry = blist_gotohash(blist->list[bucket], hash + 1, blist->bucketbits);
00526     if (entry && entry->hash == hash) {
00527         if (entry->next && (entry == blist->list[bucket])) {
00528             entry->next->prev = entry->prev;
00529             blist->list[bucket] = entry->next;
00530         } else if (entry->next) {
00531             entry->next->prev = entry->prev;
00532             entry->prev->next = entry->next;
00533         } else if (entry == blist->list[bucket]) {
00534             blist->list[bucket] = NULL;
00535         } else {
00536             entry->prev->next = NULL;
00537             blist->list[bucket]->prev = entry->prev;
```

```
00538         }
00539         objunref(entry->data->data);
00540         free(entry);
00541         objlock(blist);
00542         blist->count--;
00543         blist->version[bucket]++;
00544         objunlock(blist);
00545     }
00546     pthread_mutex_unlock(&blist->locks[bucket]);
00547 }
```

12.3.3.9 void remove_bucket_loop (struct bucket_loop * *bloop*)

Safely remove a item from a list while iterating in a loop.

While traversing the bucket list its best to use this function to remove a reference and delete it from the list.

Note

Removeing a item from the list without using this function will cause the the version to change and the iterator to rewind and fast forward.

Parameters

bloop	Bucket iterator.

Definition at line 710 of file refobj.c.

References bucket_loop::blist, bucket_loop::bucket, bucket_list::bucketbits, bucket_list::count, bucket_loop::cur, bucket_loop::cur_hash, ref_obj::data, blist_obj::data, blist_obj::hash, bucket_list::list, bucket_list::locks, blist_obj::next, objlock(), objunlock(), objunref(), blist_obj::prev, bucket_list::version, and bucket_loop::version.

```
00710                                                       {
00711     struct bucket_list *blist = bloop->blist;
00712     int bucket = bloop->bucket;
00713 
00714     pthread_mutex_lock(&blist->locks[bloop->bucket]);
00715     /*if the bucket has altered need to verify i can remove*/
00716     if (bloop->cur_hash && (!bloop->cur || (blist->version[bloop->
      bucket] != bloop->version))) {
00717         bloop->cur = blist_gotohash(blist->list[bloop->bucket], bloop->
      cur_hash + 1, blist->bucketbits);
00718         if (!bloop->cur || (bloop->cur->hash != bloop->cur_hash)) {
00719             pthread_mutex_unlock(&blist->locks[bucket]);
00720             return;
00721         }
00722     }
00723 
00724     if (!bloop->cur) {
00725         pthread_mutex_unlock(&blist->locks[bucket]);
00726         return;
00727     }
00728 
00729     if (bloop->cur->next && (bloop->cur == blist->list[bucket])) {
00730         bloop->cur->next->prev = bloop->cur->prev;
00731         blist->list[bucket] = bloop->cur->next;
00732     } else if (bloop->cur->next) {
00733         bloop->cur->next->prev = bloop->cur->prev;
00734         bloop->cur->prev->next = bloop->cur->next;
00735     } else if (bloop->cur == blist->list[bucket]) {
00736         blist->list[bucket] = NULL;
00737     } else {
00738         bloop->cur->prev->next = NULL;
00739         blist->list[bucket]->prev = bloop->cur->prev;
00740     }
00741 
00742     objunref(bloop->cur->data->data);
00743     free(bloop->cur);
00744     bloop->cur_hash = 0;
00745     bloop->cur = NULL;
00746     blist->version[bucket]++;
00747     bloop->version++;
00748     pthread_mutex_unlock(&blist->locks[bucket]);
00749 
00750     objlock(blist);
00751     blist->count--;
00752     objunlock(blist);
00753 }
```

12.4 Posix thread interface

Functions for starting and managing threads.

Files

- file thread.c

  Functions for starting and managing threads.

Data Structures

- struct thread_pvt

  thread struct used to create threads data needs to be first element
- struct threadcontainer

  Global threads data.

Typedefs

- typedef void(∗ threadcleanup)(void ∗)

  Function called after thread termination.
- typedef void ∗(∗ threadfunc)(void ∗)

  Thread function.
- typedef int(∗ threadsighandler)(int, void ∗)

  Thread signal handler function.

Enumerations

- enum thread_option_flags { THREAD_OPTION_CANCEL = 1 << 0, THREAD_OPTION_JOINABLE = 1 << 1, THREAD_OPTION_RETURN = 1 << 2 }

  Options supplied to framework_mkthread all defaults are unset.
- enum threadopt {
  TL_THREAD_NONE = 1 << 0, TL_THREAD_RUN = 1 << 1, TL_THREAD_DONE = 1 << 2, TL_THREAD_JOIN = 1 << 3,
  TL_THREAD_STOP = 1 << 4, TL_THREAD_CAN_CANCEL = 1 << 16, TL_THREAD_JOINABLE = 1 << 17, TL_THREAD_RETURN = 1 << 18 }

  Thread status a thread can be disabled by unsetting TL_THREAD_RUN.

Functions

- int framework_threadok ()

  let threads check there status.
- int startthreads (void)

  Initialise the threadlist and start manager thread.
- void stopthreads (int join)

  Signal manager to stop and cancel all running threads.
- struct thread_pvt ∗ framework_mkthread (threadfunc func, threadcleanup cleanup, threadsighandler sig_handler, void ∗data, int flags)

  create a thread result must be unreferenced
- void jointhreads (void)

  Join the manager thread.
- int thread_signal (int sig)

  Handle signal if its for me.

Variables

- struct threadcontainer * threads = NULL

  Thread control data.
- int thread_can_start = 1

  Automatically start manager thread.

12.4.1 Detailed Description

Functions for starting and managing threads.

See Also

Thread Interface The thread interface consists of a management thread managing a hashed bucket list of threads running optional clean up when done.

12.4.2 Typedef Documentation

12.4.2.1 typedef void(* threadcleanup)(void *)

Function called after thread termination.

See Also

framework_mkthread()

Parameters

data	Reference of thread data.

Definition at line 238 of file dtsapp.h.

12.4.2.2 typedef void*(* threadfunc)(void *)

Thread function.

See Also

framework_mkthread()

Parameters

data	Poinnter to reference of thread data.

Definition at line 245 of file dtsapp.h.

12.4.2.3 typedef int(* threadsighandler)(int, void *)

Thread signal handler function.

See Also

framework_mkthread()

Parameters

data	Reference of thread data.

Definition at line 252 of file dtsapp.h.

12.4.3 Enumeration Type Documentation

12.4.3.1 enum thread_option_flags

Options supplied to framework_mkthread all defaults are unset.

Note

this is shifted 16 bits limiting 16 options this maps to high 16 bits of threadopt

Enumerator

THREAD_OPTION_CANCEL Flag to enable pthread_cancel calls this is not recomended and can lead to memory leaks.

THREAD_OPTION_JOINABLE Create the the thread joinable only do this if you will be joining it cancelable threads are best detached.

THREAD_OPTION_RETURN Return reference to thread this must be unreferenced.

Definition at line 118 of file dtsapp.h.

```
00118                           {
00120         THREAD_OPTION_CANCEL        = 1 << 0,
00122         THREAD_OPTION_JOINABLE        = 1 << 1,
00124         THREAD_OPTION_RETURN        = 1 << 2
00125 };
```

12.4.3.2 enum threadopt

Thread status a thread can be disabled by unsetting TL_THREAD_RUN.

Note

bits 16-31 are useoptions see thread_option_flags

Enumerator

TL_THREAD_NONE No status.

TL_THREAD_RUN thread is marked as running

TL_THREAD_DONE thread is marked as complete

TL_THREAD_JOIN Quit when only manager is left.

Note

This flag is only valid for manager thread

TL_THREAD_STOP Quit when only manager is left.

Note

This flag is only valid for manager thread

TL_THREAD_CAN_CANCEL Flag to enable pthread_cancel calls.

TL_THREAD_JOINABLE Flag to enable pthread_cancel calls.

TL_THREAD_RETURN

Definition at line 36 of file thread.c.

```
00036                {
00038     TL_THREAD_NONE         = 1 << 0,
00040     TL_THREAD_RUN       = 1 << 1,
00042     TL_THREAD_DONE         = 1 << 2,
00045     TL_THREAD_JOIN         = 1 << 3,
00048     TL_THREAD_STOP         = 1 << 4,
00049
00051     TL_THREAD_CAN_CANCEL     = 1 << 16,
00053     TL_THREAD_JOINABLE     = 1 << 17,
00054     TL_THREAD_RETURN     = 1 << 18
00055 };
```

12.4.4 Function Documentation

12.4.4.1 struct thread_pvt* framework_mkthread (threadfunc *func*, threadcleanup *cleanup*, threadsighandler *sig_handler*, void * *data*, int *flags*)

create a thread result must be unreferenced

Note

If the manager thread has not yet started this will start the manager thread.

Warning

THREAD_OPTION_RETURN flag controls the return of this function.
Threads should periodically check the result of framework_threadok() and cleanup or use THREAD_OPTION_CANCEL

Parameters

func	Function to run thread on.
cleanup	Cleanup function to run.
sig_handler	Thread signal handler.
data	Data to pass to callbacks.
flags	Options of thread_option_flags passed

Returns

a thread structure that must be un referencend OR NULL depending on flags.

Definition at line 387 of file thread.c.

References addtobucket(), thread_pvt::cleanup, thread_pvt::data, thread_pvt::flags, thread_pvt::func, threadcontainer::list, threadcontainer::manager, objalloc(), objlock(), objref(), objunlock(), objunref(), setflag, thread_pvt::sighandler, startthreads(), testflag, thread_pvt::thr, thread_can_start, TL_THREAD_RETURN, and TL_THREAD_RUN.

Referenced by nf_ctrack_trace(), startthreads(), and unixsocket_server().

```
00387
                                  {
00388     struct thread_pvt *thread;
00389     struct threadcontainer *tc = NULL;
00390
00391     /*Grab a reference for threads in this scope start up if we can*/
00392     if (!(tc = (objref(threads)) ? threads : NULL)) {
00393         if (!thread_can_start) {
00394             return NULL;
00395         } else if (!startthreads()) {
00396             return NULL;
00397         }
00398         if (!(tc = (objref(threads)) ? threads : NULL)) {
00399             return NULL;
00400         }
00401     }
```

```
00402
00403     objlock(tc);
00404     /* dont allow threads if no manager or it not started*/
00405     if ((!tc->manager || !func) && (func != managethread)) {
00406         /*im shuting down*/
00407         objunlock(tc);
00408         objunref(tc);
00409         return NULL;
00410     } else if (!(thread = objalloc(sizeof(*thread), free_thread))) {
00411         /* could not create*/
00412         objunlock(tc);
00413         objunref(tc);
00414         return NULL;
00415     }
00416
00417     thread->data = (objref(data)) ? data : NULL;
00418     thread->flags = flags << 16;
00419     thread->cleanup = cleanup;
00420     thread->sighandler = sig_handler;
00421     thread->func = func;
00422     objunlock(tc);
00423
00424     /* start thread and check it*/
00425     if (pthread_create(&thread->thr, NULL, threadwrap, thread) || pthread_kill(thread->
      thr, 0)) {
00426         objunref(thread);
00427         objunref(tc);
00428         return NULL;
00429     }
00430
00431     /*Activate the thread it needs to be flaged to run or it will die*/
00432     objlock(tc);
00433     addtobucket(tc->list, thread);
00434     setflag(thread, TL_THREAD_RUN);
00435     objunlock(tc);
00436     objunref(tc);
00437
00438     if (testflag(thread, TL_THREAD_RETURN)) {
00439         return thread;
00440     } else {
00441         objunref(thread);
00442         return NULL;
00443     }
00444 }
```

12.4.4.2 int framework_threadok (void)

let threads check there status.

Returns

0 if the thread should terminate.

Definition at line 143 of file thread.c.

References objunref(), testflag, thread_pvt::thr, and TL_THREAD_RUN.

```
00143                                {
00144     struct thread_pvt *thr;
00145     int ret;
00146
00147     thr = get_thread_from_id();
00148     ret =(thr) ? testflag(thr, TL_THREAD_RUN) : 0;
00149     objunref(thr);
00150
00151     return ret;
00152 }
```

12.4.4.3 void jointhreads (void)

Join the manager thread.

This will be done when you have issued stopthreads and are waiting or have completed the program and want to let the threads continue. for threads to exit.

Definition at line 450 of file thread.c.

References threadcontainer::manager, objlock(), objref(), objunlock(), objunref(), setflag, thread_pvt::thr, and TL_-THREAD_JOIN.

```
00450                                {
00451     struct threadcontainer *tc;
00452
00453     tc = (objref(threads)) ? threads : NULL;
00454     if (!tc) {
00455         return;
00456     }
00457
00458     objlock(tc);
00459     if (tc->manager) {
00460         setflag(tc->manager, TL_THREAD_JOIN);
00461         objunlock(tc);
00462         pthread_join(tc->manager->thr, NULL);
00463     } else {
00464         objunlock(tc);
00465     }
00466     objunref(tc);
00467 }
```

12.4.4.4 int startthreads (void)

Initialise the threadlist and start manager thread.

Note

There is no need to call this as it will start when first thread starts.

Returns

1 On success 0 on failure.

Definition at line 268 of file thread.c.

References create_bucketlist(), framework_mkthread(), threadcontainer::list, threadcontainer::manager, objalloc(), objref(), objunref(), THREAD_OPTION_JOINABLE, and THREAD_OPTION_RETURN.

Referenced by framework_mkthread().

```
00268                                {
00269     struct threadcontainer *tc;
00270
00271     tc = (objref(threads)) ? threads : NULL;
00272
00273     if (tc) {
00274         objunref(tc);
00275         return 1;
00276     }
00277
00278     if (!(tc = objalloc(sizeof(*threads), close_threads))) {
00279         return 0;
00280     }
00281
00282     if (!tc->list && !(tc->list = create_bucketlist(4, hash_thread))) {
00283         objunref(tc);
00284         return 0;
00285     }
00286
00287     threads = tc;
00288     if (!(tc->manager = framework_mkthread(managethread, manage_clean, manager_sig
      , NULL, THREAD_OPTION_JOINABLE | THREAD_OPTION_RETURN))) {
00289         objunref(tc);
00290         return 0;
00291     }
00292
00293     return 1;
00294 }
```

12.4.4.5 void stopthreads (int *join*)

Signal manager to stop and cancel all running threads.

This should always be called at shutdown if there have been threads started.

See Also

framework_init()
FRAMEWORK_MAIN()

Parameters

join	A non zero value to join the manager thread after flaging the shutdown.

Definition at line 303 of file thread.c.

References threadcontainer::manager, objlock(), objref(), objunlock(), objunref(), setflag, thread_pvt::thr, TL_THREAD_JOIN, and TL_THREAD_STOP.

Referenced by framework_init().

```
00303                                   {
00304     struct threadcontainer *tc;
00305
00306     tc = (objref(threads)) ? threads : NULL;
00307     if (!tc) {
00308         return;
00309     }
00310
00311     objlock(tc);
00312     if (tc->manager) {
00313         setflag(tc->manager, TL_THREAD_STOP);
00314         if (join) {
00315             setflag(tc->manager, TL_THREAD_JOIN);
00316             objunlock(tc);
00317             pthread_join(tc->manager->thr, NULL);
00318         } else {
00319             objunlock(tc);
00320         }
00321     } else {
00322         objunlock(tc);
00323     }
00324     objunlock(tc);
00325     objunref(tc);
00326 }
```

12.4.4.6 int thread_signal (int *sig*)

Handle signal if its for me.

find the thread the signal was delivered to if the signal was handled returns 1 if the thread could not be handled returns -1 returns 0 if no thread is found NB sending a signal to the current thread while threads is locked will cause a deadlock.

Warning

This is not to be called directly but by the installed system signal handler.

Note

This is not supported on Win32

Parameters

sig	Signal to pass.

Returns

1 on success -1 on error.

Definition at line 496 of file thread.c.

References objunref().

```
00496                                   {
00497     int ret = 0;
00498 #ifndef __WIN32
00499     struct thread_pvt *thread = NULL;
00500
00501     if (!(thread = get_thread_from_id())) {
00502         return 0;
00503     }
00504
00505     switch(sig) {
00506         case SIGUSR1:
00507         case SIGUSR2:
00508         case SIGHUP:
00509         case SIGALRM:
00510             ret = handle_thread_signal(thread, sig);
00511             break;
00512         case SIGINT:
00513         case SIGTERM:
00514             ret = handle_thread_signal(thread, sig);
00515     }
00516     objunref(thread);
00517 #endif
00518     return ret;
00519 }
```

12.4.5 Variable Documentation

12.4.5.1 int thread_can_start = 1

Automatically start manager thread.

If threads have not been started and a thread is created the manager thread will be started. once threads have stoped this will be set to zero manually starting startthreads will be possible.

Definition at line 92 of file thread.c.

Referenced by framework_mkthread().

12.4.5.2 struct threadcontainer* threads = NULL

Thread control data.

Definition at line 86 of file thread.c.

12.5 Network socket interface

Allocate and initialise a socket for use as a client or server.

Modules

- SSL socket support

  TLSv1 SSLv2 SSLv3 DTLSv1 support.
- Unix domain sockets

  Support for unixdomain sockets using sockets interface.
- Multicast sockets

  Support for multicast sockets either IPv4 or IPv6.

Files

- file socket.c

  Allocate and initialise a socket for use as a client or server.
- file sslutil.c

  TLSv1 SSLv2 SSLv3 DTLSv1 support.

Data Structures

- union sockstruct

  Socket union describing all address types.
- struct fwsocket

  Socket data structure.
- struct socket_handler

  Socket handling thread data.

Typedefs

- typedef void(∗ socketrecv)(struct fwsocket ∗, void ∗)

  Callback function to register with a socket that will be called when there is data available.

Enumerations

- enum sock_flags {
  SOCK_FLAG_BIND = 1 << 0, SOCK_FLAG_CLOSE = 1 << 1, SOCK_FLAG_SSL = 1 << 2, SOCK_FLAG_UNIX = 1 << 3,
  SOCK_FLAG_MCAST = 1 << 4 }

  Socket flags controling a socket.

Functions

- void close_socket (struct fwsocket ∗sock)

  Mark the socket for closure and release the reference.
- struct fwsocket ∗ make_socket (int family, int type, int proto, void ∗ssl)

  Allocate a socket structure and return reference.
- struct fwsocket ∗ accept_socket (struct fwsocket ∗sock)

Create and return a socket structure from accept()

- struct fwsocket * sockconnect (int family, int stype, int proto, const char *ipaddr, const char *port, void *ssl)

  Generic client socket.
- struct fwsocket * udpconnect (const char *ipaddr, const char *port, void *ssl)

  UDP Socket client.
- struct fwsocket * tcpconnect (const char *ipaddr, const char *port, void *ssl)

  TCP Socket client.
- struct fwsocket * sockbind (int family, int stype, int proto, const char *ipaddr, const char *port, void *ssl, int backlog)

  Generic server socket.
- struct fwsocket * udpbind (const char *ipaddr, const char *port, void *ssl)

  UDP server socket.
- struct fwsocket * tcpbind (const char *ipaddr, const char *port, void *ssl, int backlog)

  Generic server socket.
- void socketserver (struct fwsocket *sock, socketrecv read, socketrecv acceptfunc, threadcleanup cleanup, void *data)

  Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.
- void socketclient (struct fwsocket *sock, void *data, socketrecv read, threadcleanup cleanup)

  Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.
- const char * sockaddr2ip (union sockstruct *addr, char *buff, int blen)

  Return the ip address of a sockstruct addr.
- int socketread_d (struct fwsocket *sock, void *buf, int num, union sockstruct *addr)

  Read from a socket into a buffer.
- int socketread (struct fwsocket *sock, void *buf, int num)

  Read from a socket into a buffer.
- int socketwrite_d (struct fwsocket *sock, const void *buf, int num, union sockstruct *addr)

  Write a buffer to a socket.
- int socketwrite (struct fwsocket *sock, const void *buf, int num)

  Write a buffer to a socket.

12.5.1 Detailed Description

Allocate and initialise a socket for use as a client or server.

See Also

Socket Interface
Socket Example (Echo Server/Client)

12.5.2 Typedef Documentation

12.5.2.1 typedef void(* socketrecv)(struct fwsocket *, void *)

Callback function to register with a socket that will be called when there is data available.

Parameters

sock	Socket structure data arrived on.
data	Reference to data held by client/server thread.

Definition at line 259 of file dtsapp.h.

12.5.3 Enumeration Type Documentation

12.5.3.1 enum sock_flags

Socket flags controling a socket.

Enumerator

SOCK_FLAG_BIND The socket has been bound and awaiting connections.

SOCK_FLAG_CLOSE The socket is going away stop processing in its thread.

SOCK_FLAG_SSL SSL has been requested on this socket dont allow clear read/send.

SOCK_FLAG_UNIX UNIX Domain Socket.

SOCK_FLAG_MCAST Multicast Socket.

Definition at line 102 of file dtsapp.h.

```
00102                 {
00104     SOCK_FLAG_BIND        = 1 << 0,
00106     SOCK_FLAG_CLOSE      = 1 << 1,
00108     SOCK_FLAG_SSL       = 1 << 2,
00110     SOCK_FLAG_UNIX        = 1 << 3,
00112     SOCK_FLAG_MCAST      = 1 << 4
00113 };
```

12.5.4 Function Documentation

12.5.4.1 struct fwsocket* accept_socket (struct fwsocket * *sock*)

Create and return a socket structure from accept()

Parameters

sock	Reference to the socket its accepted on.

Returns

Reference to new socket.

Definition at line 144 of file socket.c.

References fwsocket::addr, objalloc(), objlock(), objunlock(), objunref(), fwsocket::proto, sockstruct::sa, fwsocket::sock, fwsocket::ssl, tlsaccept(), and fwsocket::type.

```
00144                                                                {
00145     struct fwsocket *si;
00146     socklen_t salen = sizeof(si->addr);
00147
00148     if (!(si = objalloc(sizeof(*si),clean_fwsocket))) {
00149         return NULL;
00150     }
00151
00152     objlock(sock);
00153     if ((si->sock = accept(sock->sock, &si->addr.sa, &salen)) < 0) {
00154         objunlock(sock);
00155         objunref(si);
00156         return NULL;
00157     }
00158
00159     si->type = sock->type;
00160     si->proto = sock->proto;
00161
00162     if (sock->ssl) {
00163         tlsaccept(si, sock->ssl);
00164     }
00165     objunlock(sock);
00166
00167     return (si);
00168 }
```

12.5.4.2 void close_socket (struct fwsocket * *sock*)

Mark the socket for closure and release the reference.

Parameters

sock	Socket to close.

Definition at line 79 of file socket.c.

References objunref(), setflag, and SOCK_FLAG_CLOSE.

Referenced by socktest(), and unixsocktest().

```
00079                                                   {
00080     if (sock) {
00081         setflag(sock, SOCK_FLAG_CLOSE);
00082         objunref(sock);
00083     }
00084 }
```

12.5.4.3 struct fwsocket* make_socket (int *family*, int *type*, int *proto*, void * *ssl*)

Allocate a socket structure and return reference.

The socket FD is assined by a call to socket.

Warning

This function should not be called directly.

Parameters

family	Protocol family.
type	Socket type.
proto	Protocol to be used.
ssl	SSL structure to associate with the socket.

Returns

Reference to socket structure holding a FD.

Definition at line 120 of file socket.c.

References objalloc(), objunref(), fwsocket::proto, fwsocket::sock, fwsocket::ssl, and fwsocket::type.

Referenced by dtls_listenssl(), mcast_socket(), unixsocket_client(), and unixsocket_server().

```
00120                                                                                   {
00121     struct fwsocket *si;
00122
00123     if (!(si = objalloc(sizeof(*si),clean_fwsocket))) {
00124         return NULL;
00125     }
00126
00127     if ((si->sock = socket(family, type, proto)) < 0) {
00128         objunref(si);
00129         return NULL;
00130     };
00131
00132     if (ssl) {
00133         si->ssl = ssl;
00134     }
00135     si->type = type;
00136     si->proto = proto;
00137
00138     return (si);
00139 }
```

12.5.4.4 const char* sockaddr2ip (union sockstruct * *addr*, char * *buff*, int *blen*)

Return the ip address of a sockstruct addr.

Parameters

addr	Socketstruct to return the address for.
buff	Buffer the IP will be copied too.
blen	Buffer length.

Returns

a pointer to buff.

Definition at line 504 of file socket.c.

References inet_ntop(), sockstruct::sa4, sockstruct::sa6, and sockstruct::ss.

```
00504                                                                   {
00505     if (!buff) {
00506         return NULL;
00507     }
00508
00509     switch (addr->ss.ss_family) {
00510         case PF_INET:
00511             inet_ntop(PF_INET, &addr->sa4.sin_addr, buff, blen);
00512             break;
00513         case PF_INET6:
00514             inet_ntop(PF_INET6, &addr->sa6.sin6_addr, buff, blen);
00515             break;
00516     }
00517     return buff;
00518 }
```

12.5.4.5 struct fwsocket* sockbind (int *family*, int *stype*, int *proto*, const char * *ipaddr*, const char * *port*, void * *ssl*, int *backlog*)

Generic server socket.

See Also

udpbind
tcpbind

Parameters

family	Protocol family.
stype	Socket type.
proto	Socket protocol.
ipaddr	Ipaddr to connect too.
port	Port to connect too.
ssl	SSL structure to associate with socket.
backlog	Connection backlog passed to listen.

Returns

Reference to socket structure.

Definition at line 290 of file socket.c.

```
00290
                                {
00291     return(_opensocket(family, stype, proto, ipaddr, port, ssl, 1, backlog));
00292 }
```

12.5.4.6 struct fwsocket* sockconnect (int *family*, int *stype*, int *proto*, const char * *ipaddr*, const char * *port*, void * *ssl*)

Generic client socket.

See Also

udpconnect
tcpconnect

Parameters

family	Protocol family.
stype	Socket type.
proto	Socket protocol.
ipaddr	Ipaddr to connect too.
port	Port to connect too.
ssl	SSL structure to associate with socket.

Returns

Reference to socket structure.

Definition at line 250 of file socket.c.

```
00250                     {
00251     return(_opensocket(family, stype, proto, ipaddr, port, ssl, 0, 0));
00252 }
```

12.5.4.7 void socketclient (struct fwsocket * *sock*, void * *data*, socketrecv *read*, threadcleanup *cleanup*)

Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.

See Also

sockclient
threadcleanup
socketrecv

Parameters

sock	Reference to a bound socket.
data	to send to the callbacks in paramaters.
read	Callback to handle data when ready to read.
cleanup	Thread cleanup function for when the socket closes.

Definition at line 493 of file socket.c.

References startsslclient().

Referenced by socktest().

```
00493                                                                                                    {
00494     startsslclient(sock);
00495
00496     _start_socket_handler(sock, read, NULL, cleanup, data);
00497 }
```

12.5.4.8 int socketread (struct fwsocket * *sock*, void * *buf*, int *num*)

Read from a socket into a buffer.

There are 2 functions each for reading and writing data to a socket.

Connected (client) sockets (Including UDP [SOCK_DGRAM])

Use of socketwrite and socketread is acceptable.

UDP (SOCK_DGRAM) servers.

These require use of socketread_d and socketwrite_d the exception is DTLS connections that use there own routines and either works.

Parameters

sock	Socket structure to read from.
buf	Buffer to fill.
num	Size of the buffer.

Returns

Number of bytes read or -1 on error 0 will indicate connection closed.

Definition at line 489 of file sslutil.c.

References socketread_d().

Referenced by client_func().

```
00489                                                                   {
00490     return (socketread_d(sock, buf, num, NULL));
00491 }
```

12.5.4.9 int socketread_d (struct fwsocket * *sock*, void * *buf*, int *num*, union sockstruct * *addr*)

Read from a socket into a buffer.

There are 2 functions each for reading and writing data to a socket.

Connected (client) sockets (Including UDP [SOCK_DGRAM])

Use of socketwrite and socketread is acceptable.

UDP (SOCK_DGRAM) servers.

These require use of socketread_d and socketwrite_d the exception is DTLS connections that use there own routines and either works.

Parameters

sock	Socket structure to read from.
buf	Buffer to fill.
num	Size of the buffer.
addr	Addr structure to fill remote address in.

Returns

Number of bytes read or -1 on error 0 will indicate connection closed.

Definition at line 406 of file sslutil.c.

References fwsocket::flags, objlock(), objunlock(), objunref(), sockstruct::sa, fwsocket::sock, SOCK_FLAG_CLOSE, SOCK_FLAG_SSL, ssldata::ssl, fwsocket::ssl, testflag, and fwsocket::type.

Referenced by server_func(), and socketread().

```
00406                                                                                                {
00407     struct ssldata *ssl = sock->ssl;
00408     socklen_t salen = sizeof(*addr);
00409     int ret, err, syserr;
00410 
00411     if (!ssl && !testflag(sock, SOCK_FLAG_SSL)) {
00412         objlock(sock);
00413         if (addr && (sock->type == SOCK_DGRAM)) {
00414             ret = recvfrom(sock->sock, buf, num, 0, &addr->sa, &salen);
00415         } else {
00416 #ifndef __WIN32
00417             ret = read(sock->sock, buf, num);
00418 #else
00419             ret = recv(sock->sock, buf, num, 0);
00420 #endif
00421         }
00422         if (ret == 0) {
00423             sock->flags |= SOCK_FLAG_CLOSE;
00424         }
00425         objunlock(sock);
00426         return (ret);
00427     } else if (!ssl) {
00428         return -1;
00429     }
00430 
00431     objlock(ssl);
00432     /* ive been shutdown*/
00433     if (!ssl->ssl) {
00434         objunlock(ssl);
00435         return (-1);
00436     }
00437     ret = SSL_read(ssl->ssl, buf, num);
00438     err = SSL_get_error(ssl->ssl, ret);
00439     if (ret == 0) {
00440         sock->flags |= SOCK_FLAG_CLOSE;
00441     }
00442     objunlock(ssl);
00443     switch (err) {
00444         case SSL_ERROR_NONE:
00445             break;
00446         case SSL_ERROR_WANT_X509_LOOKUP:
00447             printf("Want X509\n");
00448             break;
00449         case SSL_ERROR_WANT_READ:
00450             printf("Read Want Read\n");
00451             break;
00452         case SSL_ERROR_WANT_WRITE:
00453             printf("Read Want write\n");
00454             break;
00455         case SSL_ERROR_ZERO_RETURN:
00456         case SSL_ERROR_SSL:
00457             objlock(sock);
00458             objunref(sock->ssl);
00459             sock->ssl = NULL;
00460             objunlock(sock);
00461             break;
00462         case SSL_ERROR_SYSCALL:
00463             syserr = ERR_get_error();
00464             if (syserr || (!syserr && (ret == -1))) {
00465                 printf("R syscall %i %i\n", syserr, ret);
00466             }
00467             break;
00468         default
00469                 :
00470             printf("other\n");
00471             break;
00472     }
00473 
00474     return (ret);
00475 }
```

12.5.4.10 void socketserver (struct fwsocket * *sock*, socketrecv *read*, socketrecv *acceptfunc*, threadcleanup *cleanup*, void * *data*)

Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.

See Also

sockclient
threadcleanup
socketrecv

Parameters

sock	Reference to a bound socket.
read	Callback to handle data when ready to read.
acceptfunc	Function to call on connection accept.
cleanup	Thread cleanup function for when the socket closes.
data	to send to the callbacks in paramaters.

Definition at line 463 of file socket.c.

References fwsocket::children, create_bucketlist(), dtsl_serveropts(), fwsocket::flags, objlock(), objunlock(), fwsocket::ssl, and fwsocket::type.

Referenced by socktest().

```
00464                                                                                {
00465
00466     objlock(sock);
00467     if (sock->flags & SOCK_FLAG_BIND) {
00468         if (sock->ssl || !(sock->type == SOCK_DGRAM)) {
00469             sock->children = create_bucketlist(6, hash_socket);
00470         }
00471         if (sock->ssl && (sock->type == SOCK_DGRAM)) {
00472             objunlock(sock);
00473             dtsl_serveropts(sock);
00474         } else {
00475             objunlock(sock);
00476         }
00477     } else {
00478         objunlock(sock);
00479     }
00480     _start_socket_handler(sock, read, acceptfunc, cleanup, data);
00481 }
```

12.5.4.11 int socketwrite (struct fwsocket * *sock*, const void * *buf*, int *num*)

Write a buffer to a socket.

There are 2 functions each for reading and writing data to a socket.

Connected (client) sockets (Including UDP [SOCK_DGRAM])

Use of socketwrite and socketread is acceptable.

UDP (SOCK_DGRAM) servers.

These require use of socketread_d and socketwrite_d the exception is DTLS connections that use there own routines and either works.

Parameters

sock	Socket structure to send data too.
buf	Buffer to send.
num	Lengthe of the buffer.

Returns

Number of bytes written or -1 on error 0 will indicate some error in SSL.

Definition at line 629 of file sslutil.c.

References socketwrite_d().

Referenced by client_func(), and socktest().

```
00629                                                                   {
00630     return (socketwrite_d(sock, buf, num, NULL));
00631 }
```

12.5.4.12 int socketwrite_d (struct fwsocket * *sock*, const void * *buf*, int *num*, union sockstruct * *addr*)

Write a buffer to a socket.

There are 2 functions each for reading and writing data to a socket.

Connected (client) sockets (Including UDP [SOCK_DGRAM])

Use of socketwrite and socketread is acceptable.

UDP (SOCK_DGRAM) servers.

These require use of socketread_d and socketwrite_d the exception is DTLS connections that use there own routines and either works.

Todo implement send/sendto in WIN32

Parameters

sock	Socket structure to send data too.
buf	Buffer to send.
num	Lengthe of the buffer.
addr	Addr structure to send the buffer too (SOCK_DGRAM) see notes.

Returns

Number of bytes written or -1 on error 0 will indicate some error in SSL.

Definition at line 508 of file sslutil.c.

References fwsocket::addr, fwsocket::flags, objlock(), objunlock(), objunref(), sockstruct::sa, setflag, fwsocket::sock, SOCK_FLAG_CLOSE, SOCK_FLAG_MCAST, SOCK_FLAG_SSL, SOCK_FLAG_UNIX, sockstruct::ss, ssldata::ssl, fwsocket::ssl, testflag, fwsocket::type, and sockstruct::un.

Referenced by server_func(), socketwrite(), and unixsocktest().

```
00508                                                                                                   {
00509     struct ssldata *ssl = (sock) ? sock->ssl : NULL;
00510     int ret, err, syserr;
00511
00512     if (!sock) {
00513         return (-1);
00514     }
00515
00516     if (!ssl && !testflag(sock, SOCK_FLAG_SSL)) {
```

```
00517         objlock(sock);
00518         if (addr && (sock->type == SOCK_DGRAM)) {
00519 #ifndef __WIN32
00520             if (sock->flags & SOCK_FLAG_UNIX) {
00521                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, (const struct sockaddr *)&addr->
      un, sizeof(addr->un));
00522             } else if (sock->flags & SOCK_FLAG_MCAST) {
00523                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &sock->addr.
      sa, sizeof(sock->addr.ss));
00524             } else {
00525                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &addr->sa, sizeof(*addr));
00526             }
00527 #else
00528             if (sock->flags & SOCK_FLAG_MCAST) {
00529                 ret = sendto(sock->sock, buf, num, 0, &sock->addr.sa, sizeof(sock->
      addr.ss));
00530             } else {
00531                 ret = sendto(sock->sock, buf, num, 0, &addr->sa, sizeof(*addr));
00532             }
00533 #endif
00534         } else {
00535 #ifndef __WIN32
00536             if (sock->flags & SOCK_FLAG_MCAST) {
00537                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &sock->addr.
      sa, sizeof(sock->addr.ss));
00538             } else {
00539                 ret = send(sock->sock, buf, num, MSG_NOSIGNAL);
00540             }
00541 #else
00542             if (sock->flags & SOCK_FLAG_MCAST) {
00543                 ret = sendto(sock->sock, buf, num, 0, &sock->addr.sa, sizeof(sock->
      addr.ss));
00544             } else {
00545                 ret = send(sock->sock, buf, num, 0);
00546             }
00547 #endif
00548         }
00549         if (ret == -1) {
00550             switch(errno) {
00551                 case EBADF:
00552                 case EPIPE:
00553 #ifndef __WIN32
00554                 case ENOTCONN:
00555                 case ENOTSOCK:
00556 #endif
00557                     sock->flags |= SOCK_FLAG_CLOSE;
00558                     break;
00559             }
00560         }
00561         objunlock(sock);
00562         return (ret);
00563     } else if (!ssl) {
00564         return -1;
00565     }
00566 
00567     if (ssl && ssl->ssl) {
00568         objlock(ssl);
00569         if (SSL_state(ssl->ssl) != SSL_ST_OK) {
00570             objunlock(ssl);
00571             return (SSL_ERROR_SSL);
00572         }
00573         ret = SSL_write(ssl->ssl, buf, num);
00574         err = SSL_get_error(ssl->ssl, ret);
00575         objunlock(ssl);
00576     } else {
00577         return -1;
00578     }
00579 
00580     if (ret == -1) {
00581         setflag(sock, SOCK_FLAG_CLOSE);
00582     }
00583 
00584     switch(err) {
00585         case SSL_ERROR_NONE:
00586             break;
00587         case SSL_ERROR_WANT_READ:
00588             printf("Send Want Read\n");
00589             break;
00590         case SSL_ERROR_WANT_WRITE:
00591             printf("Send Want write\n");
00592             break;
00593         case SSL_ERROR_WANT_X509_LOOKUP:
00594             printf("Want X509\n");
00595             break;
00596         case SSL_ERROR_ZERO_RETURN:
00597         case SSL_ERROR_SSL:
00598             objlock(sock);
```

```
00599             objunref(sock->ssl);
00600             sock->ssl = NULL;
00601             objunlock(sock);
00602             break;
00603         case SSL_ERROR_SYSCALL:
00604             syserr = ERR_get_error();
00605             if (syserr || (!syserr && (ret == -1))) {
00606                 printf("W syscall %i %i\n", syserr, ret);
00607             }
00608             break;
00609         default:
00610             printf("other\n");
00611             break;
00612     }
00613
00614     return (ret);
00615 }
```

12.5.4.13 struct fwsocket* tcpbind (const char * *ipaddr*, const char * *port*, void * *ssl*, int *backlog*)

Generic server socket.

See Also

udpbind
sockbind

Parameters

ipaddr	Ipaddr to connect too.
port	Port to connect too.
ssl	SSL structure to associate with socket.
backlog	Connection backlog passed to listen.

Returns

Reference to socket structure.

Definition at line 315 of file socket.c.

Referenced by socktest().

```
00315                                                                                 {
00316     return (_opensocket(PF_UNSPEC, SOCK_STREAM, IPPROTO_TCP, ipaddr, port, ssl, 1, backlog));
00317 }
```

12.5.4.14 struct fwsocket* tcpconnect (const char * *ipaddr*, const char * *port*, void * *ssl*)

TCP Socket client.

See Also

sockconnect
udpconnect

Parameters

ipaddr	Ipaddr to connect too.
port	Port to connect too.
ssl	SSL structure to associate with socket.

Returns

Reference to socket structure.

Definition at line 274 of file socket.c.

Referenced by socktest().

```
00274                                                                                {
00275     return (_opensocket(PF_UNSPEC, SOCK_STREAM, IPPROTO_TCP, ipaddr, port, ssl, 0, 0));
00276 }
```

12.5.4.15 struct fwsocket* udpbind (const char * *ipaddr*, const char * *port*, void * *ssl*)

UDP server socket.

See Also

sockbind
tcpbind

Parameters

ipaddr	Ipaddr to connect too.
port	Port to connect too.
ssl	SSL structure to associate with socket.

Returns

Reference to socket structure.

Definition at line 302 of file socket.c.

Referenced by socktest().

```
00302                                                                                {
00303     return (_opensocket(PF_UNSPEC, SOCK_DGRAM, IPPROTO_UDP, ipaddr, port, ssl, 1, 0));
00304 }
```

12.5.4.16 struct fwsocket* udpconnect (const char * *ipaddr*, const char * *port*, void * *ssl*)

UDP Socket client.

See Also

sockconnect
tcpconnect

Parameters

ipaddr	Ipaddr to connect too.
port	Port to connect too.
ssl	SSL structure to associate with socket.

Returns

Reference to socket structure.

Definition at line 262 of file socket.c.

Referenced by socktest().

```
00262                                                                                {
00263     return (_opensocket(PF_UNSPEC, SOCK_DGRAM, IPPROTO_UDP, ipaddr, port, ssl, 0, 0));
00264 }
```

12.6 SSL socket support

TLSv1 SSLv2 SSLv3 DTLSv1 support.

Files

- file sslutil.c

  TLSv1 SSLv2 SSLv3 DTLSv1 support.

Data Structures

- struct ssldata

  SSL data structure for enabling encryption on sockets.

Macros

- #define COOKIE_SECRET_LENGTH 32

  length of cookie secret using SHA2-256 HMAC

Typedefs

- typedef struct ssldata ssldata

  Forward decleration of structure.

Enumerations

- enum SSLFLAGS {
  SSL_TLSV1 = 1 << 0, SSL_SSLV2 = 1 << 1, SSL_SSLV3 = 1 << 2, SSL_DTLSV1 = 1 << 3,
  SSL_CLIENT = 1 << 4, SSL_SERVER = 1 << 5, SSL_DTLSCON = 1 << 6 }

  SSL configuration flags.

Functions

- void ssl_shutdown (void *data, int sock)

  Shutdown the SSL connection.
- void * tlsv1_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for TLSv1.
- void * sslv2_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for SSLv2 (If available)
- void * sslv3_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for SSLv3.
- void * dtlsv1_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for DTLSv1.
- void tlsaccept (struct fwsocket *sock, struct ssldata *orig)

  Create SSL session for new connection.
- void sslstartup (void)

  Initialise SSL support this should be called at startup.
- void dtsl_serveropts (struct fwsocket *sock)

  Start up the DTLSv1 Server.

- struct fwsocket * dtls_listenssl (struct fwsocket *sock)

  Implementation of "listen" for DTLSv1.
- void startsslclient (struct fwsocket *sock)

  Start SSL on a client socket.
- void dtlstimeout (struct fwsocket *sock, struct timeval *timeleft, int defusec)

  Get DTLSv1 timeout setting todefault timeout.
- void dtlshandltimeout (struct fwsocket *sock)

  Handle DTLSv1 timeout.

12.6.1 Detailed Description

TLSv1 SSLv2 SSLv3 DTLSv1 support.

See Also

LIB-Sock This is part of the socket interface to support encrypted sockets a ssldata refernece will be created and passed on socket initialization.

This is part of the socket interface to upport encrypted sockets a ssldata refernece will be created and passed on socket initialization.

See Also

Network socket interface

12.6.2 Macro Definition Documentation

12.6.2.1 #define COOKIE_SECRET_LENGTH 32

length of cookie secret using SHA2-256 HMAC

Definition at line 83 of file sslutil.c.

Referenced by sslstartup().

12.6.3 Typedef Documentation

12.6.3.1 typedef struct ssldata ssldata

Forward decleration of structure.

Definition at line 97 of file dtsapp.h.

12.6.4 Enumeration Type Documentation

12.6.4.1 enum SSLFLAGS

SSL configuration flags.

Enumerator

SSL_TLSV1 TLSv1.

SSL_SSLV2 SSLv2 This may not be available due to security issues.

SSL_SSLV3 SSLv3.

SSL_DTLSV1 DTLSv1 (UDP Connections)

SSL_CLIENT This session is client mode.

SSL_SERVER This session is server mode.

SSL_DTLSCON UDP connection is listening.

Definition at line 48 of file sslutil.c.

```
00048                {
00050     SSL_TLSV1   = 1 << 0,
00052     SSL_SSLV2   = 1 << 1,
00054     SSL_SSLV3   = 1 << 2,
00056     SSL_DTLSV1    = 1 << 3,
00058     SSL_CLIENT    = 1 << 4,
00060     SSL_SERVER    = 1 << 5,
00062     SSL_DTLSCON   = 1 << 6
00063 };
```

12.6.5 Function Documentation

12.6.5.1 struct fwsocket* dtls_listenssl (struct fwsocket * *sock*)

Implementation of "listen" for DTLSv1.

Warning

Do not call this directly.

Parameters

sock	Reference to server socket.

Returns

New socket reference for the new connection.

Definition at line 731 of file sslutil.c.

References fwsocket::addr, ssldata::flags, make_socket(), objalloc(), objlock(), objunlock(), objunref(), fwsocket::proto, sockstruct::sa, setflag, fwsocket::sock, SOCK_FLAG_SSL, ssldata::ssl, fwsocket::ssl, SSL_DTLSCON, and fwsocket::type.

```
00731                                                                {
00732     struct ssldata *ssl = sock->ssl;
00733     struct ssldata *newssl;
00734     struct fwsocket *newsock;
00735     union sockstruct client;
00736 #ifndef __WIN32__
00737     int on = 1;
00738 #else
00739 /*  unsigned long on = 1;*/
00740 #endif
00741
00742     if (!(newssl = objalloc(sizeof(*newssl), free_ssldata))) {
00743         return NULL;
00744     }
00745
00746     newssl->flags |= SSL_DTLSCON;
00747
00748     dtlssetopts(newssl, ssl, sock);
00749     memset(&client, 0, sizeof(client));
00750     if (DTLSv1_listen(newssl->ssl, &client) <= 0) {
00751         objunref(newssl);
00752         return NULL;
00753     }
00754
00755     objlock(sock);
00756     if (!(newsock = make_socket(sock->addr.sa.sa_family, sock->
    type, sock->proto, newssl))) {
00757         objunlock(sock);
00758         objunref(newssl);
00759         return NULL;
```

```
00760     }
00761     objunlock(sock);
00762     memcpy(&newsock->addr, &client, sizeof(newsock->addr));
00763 #ifndef __WIN32__
00764     setsockopt(newsock->sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
00765 #ifdef SO_REUSEPORT
00766     setsockopt(newsock->sock, SOL_SOCKET, SO_REUSEPORT, &on, sizeof(on));
00767 #endif
00768 #else
00769 /*  ioctlsocket(newsock->sock, FIONBIO, (unsigned long*)&on);*/
00770 #endif
00771     objlock(sock);
00772     bind(newsock->sock, &sock->addr.sa, sizeof(sock->addr));
00773     objunlock(sock);
00774     connect(newsock->sock, &newsock->addr.sa, sizeof(newsock->addr));
00775
00776     dtlsaccept(newsock);
00777     setflag(newsock, SOCK_FLAG_SSL);
00778
00779     return (newsock);
00780 }
```

12.6.5.2 void dtlshandltimeout (struct fwsocket * *sock*)

Handle DTLSv1 timeout.

Parameters

sock	Reference to socket.

Definition at line 846 of file sslutil.c.

References objlock(), objunlock(), ssldata::ssl, and fwsocket::ssl.

```
00846                                                 {
00847     if (!sock->ssl) {
00848         return;
00849     }
00850
00851     objlock(sock->ssl);
00852     DTLSv1_handle_timeout(sock->ssl->ssl);
00853     objunlock(sock->ssl);
00854 }
```

12.6.5.3 void dtlstimeout (struct fwsocket * *sock*, struct timeval * *timeleft*, int *defusec*)

Get DTLSv1 timeout setting todefault timeout.

Warning

Do not call this directly.

Parameters

sock	Reference to socket.
timeleft	timeval to store timeleft or set to default.
defusec	Default timeout to set.

Definition at line 831 of file sslutil.c.

References objlock(), objunlock(), ssldata::ssl, and fwsocket::ssl.

```
00831                                                                   {
00832     if (!sock || !sock->ssl || !sock->ssl->ssl) {
00833         return;
00834     }
00835
00836     objlock(sock->ssl);
00837     if (!DTLSv1_get_timeout(sock->ssl->ssl, timeleft)) {
00838         timeleft->tv_sec = 0;
```

```
00839         timeleft->tv_usec = defusec;
00840     }
00841     objunlock(sock->ssl);
00842 }
```

12.6.5.4 void* dtlsv1_init (const char * *cacert*, const char * *cert*, const char * *key*, int *verify*)

Create a SSL structure for DTLSv1.

Parameters

cacert	Path to the CA certificate[s].
cert	Public certificate to use.
key	Private key file.
verify	OpenSSL flags.

Definition at line 325 of file sslutil.c.

References ssldata::ctx, ssldata::ssl, and SSL_DTLSV1.

Referenced by socktest().

```
00325                                                                                                {
00326     const SSL_METHOD *meth = DTLSv1_method();
00327     struct ssldata *ssl;
00328
00329     ssl = sslinit(cacert, cert, key, verify, meth, SSL_DTLSV1);
00330     /* XXX BIO_CTRL_DGRAM_MTU_DISCOVER*/
00331     SSL_CTX_set_read_ahead(ssl->ctx, 1);
00332
00333     return (ssl);
00334 }
```

12.6.5.5 void dtsl_serveropts (struct fwsocket * *sock*)

Start up the DTLSv1 Server.

Warning

This should not be called directly

See Also

socketserver

Parameters

sock	Reference to socket structure of DTLSv1 Server

Definition at line 685 of file sslutil.c.

References ssldata::ctx, ssldata::flags, objlock(), objunlock(), ssldata::ssl, fwsocket::ssl, and SSL_SERVER.

Referenced by socketserver().

```
00685                                                {
00686     struct ssldata *ssl = sock->ssl;
00687
00688     if (!ssl) {
00689         return;
00690     }
00691
00692     dtlssetopts(ssl, NULL, sock);
00693
00694     objlock(ssl);
00695     SSL_CTX_set_cookie_generate_cb(ssl->ctx, generate_cookie);
00696     SSL_CTX_set_cookie_verify_cb(ssl->ctx, verify_cookie);
```

```
00697     SSL_CTX_set_session_cache_mode(ssl->ctx, SSL_SESS_CACHE_OFF);
00698
00699     SSL_set_options(ssl->ssl, SSL_OP_COOKIE_EXCHANGE);
00700     ssl->flags |= SSL_SERVER;
00701     objunlock(ssl);
00702 }
```

12.6.5.6 void ssl_shutdown (void * *data*, int *sock*)

Shutdown the SSL connection.

Extra read/write may be required if so use select on failure the port has probably gone only try 3 times.

Todo Make sure this is only called when the thread has stoped selecting here may be wrong.

Parameters

data	Refernece to the SSL data of socket.
sock	Socket FD to wait for data on.

Definition at line 179 of file sslutil.c.

References objlock(), objunlock(), and ssldata::ssl.

```
00179                                                {
00180     struct ssldata *ssl = data;
00181     int ret, selfd, cnt = 0;
00182
00183     if (!ssl) {
00184         return;
00185     }
00186
00187     objlock(ssl);
00188
00189     while (ssl->ssl &&  (ret = _ssl_shutdown(ssl) && (cnt < 3))) {
00190         selfd = socket_select(sock, ret);
00191         if (selfd <= 0) {
00192             break;
00193         }
00194         cnt++;
00195     }
00196
00197     if (ssl->ssl) {
00198         SSL_free(ssl->ssl);
00199         ssl->ssl = NULL;
00200     }
00201     objunlock(ssl);
00202 }
```

12.6.5.7 void sslstartup (void)

Initialise SSL support this should be called at startup.

See Also

FRAMEWORK_MAIN

Definition at line 639 of file sslutil.c.

References COOKIE_SECRET_LENGTH, and genrand().

Referenced by framework_init().

```
00639                              {
00640     SSL_library_init();
00641     SSL_load_error_strings();
00642     OpenSSL_add_ssl_algorithms();
00643
00644     if ((cookie_secret = malloc(COOKIE_SECRET_LENGTH))) {
00645         genrand(cookie_secret, COOKIE_SECRET_LENGTH);
00646     }
00647 }
```

12.6.5.8 void* sslv2_init (const char * *cacert*, const char * *cert*, const char * *key*, int *verify*)

Create a SSL structure for SSLv2 (If available)

Parameters

cacert	Path to the CA certificate[s].
cert	Public certificate to use.
key	Private key file.
verify	OpenSSL flags.

Definition at line 299 of file sslutil.c.

References SSL_SSLV2.

```
00299                                                                                {
00300     const SSL_METHOD *meth = SSLv2_method();
00301
00302     return (sslinit(cacert, cert, key, verify, meth, SSL_SSLV2));
00303 }
```

12.6.5.9 void* sslv3_init (const char * *cacert*, const char * *cert*, const char * *key*, int *verify*)

Create a SSL structure for SSLv3.

Parameters

cacert	Path to the CA certificate[s].
cert	Public certificate to use.
key	Private key file.
verify	OpenSSL flags.

Definition at line 311 of file sslutil.c.

References ssldata::ssl, and SSL_SSLV3.

Referenced by socktest().

```
00311                                                                                {
00312     const SSL_METHOD *meth = SSLv3_method();
00313     struct ssldata *ssl;
00314
00315     ssl = sslinit(cacert, cert, key, verify, meth, SSL_SSLV3);
00316
00317     return (ssl);
00318 }
```

12.6.5.10 void startsslclient (struct fwsocket * *sock*)

Start SSL on a client socket.

Warning

This should not be called directly

See Also

clientsocket()

Parameters

sock	Reference to client socket.

Definition at line 811 of file sslutil.c.

References ssldata::flags, fwsocket::ssl, SSL_SERVER, and fwsocket::type.

Referenced by socketclient().

```
00811                                                   {
00812     if (!sock || !sock->ssl || (sock->ssl->flags & SSL_SERVER)) {
00813         return;
00814     }
00815 
00816     switch(sock->type) {
00817         case SOCK_DGRAM:
00818             dtlsconnect(sock);
00819             break;
00820         case SOCK_STREAM:
00821             sslsockstart(sock, NULL, 0);
00822             break;
00823     }
00824 }
```

12.6.5.11 void tlsaccept (struct fwsocket * *sock*, struct ssldata * *orig*)

Create SSL session for new connection.

Warning

This should never be called.

Parameters

sock	Reference too new incoming socket.
orig	Servers SSL session to clone.

Definition at line 382 of file sslutil.c.

References objalloc(), setflag, SOCK_FLAG_SSL, and fwsocket::ssl.

Referenced by accept_socket().

```
00382                                                                   {
00383     setflag(sock, SOCK_FLAG_SSL);
00384     if ((sock->ssl = objalloc(sizeof(*sock->ssl), free_ssldata))) {
00385         sslsockstart(sock, orig, 1);
00386     }
00387 }
```

12.6.5.12 void* tlsv1_init (const char * *cacert*, const char * *cert*, const char * *key*, int *verify*)

Create a SSL structure for TLSv1.

Parameters

cacert	Path to the CA certificate[s].
cert	Public certificate to use.
key	Private key file.
verify	OpenSSL flags.

Definition at line 287 of file sslutil.c.

References SSL_TLSV1.

```
00287                                                                                                   {
00288     const SSL_METHOD *meth = TLSv1_method();
00289
00290     return (sslinit(cacert, cert, key, verify, meth, SSL_TLSV1));
00291 }
```

12.7 Unix domain sockets

Support for unixdomain sockets using sockets interface.

Files

- file unixsock.c

  Attach a thread to a unix socket start a new thread on connect.

Data Structures

- struct unixserv_sockthread

  Unix socket server data structure.
- struct unixclient_sockthread

  Unix socket client data structure.

Functions

- struct fwsocket ∗ unixsocket_server (const char ∗sock, int protocol, int mask, socketrecv read, void ∗data)

  Create and run UNIX server socket thread.
- struct fwsocket ∗ unixsocket_client (const char ∗sock, int protocol, socketrecv read, void ∗data)

  Create a client thread on the socket.

12.7.1 Detailed Description

Support for unixdomain sockets using sockets interface. A thread is started on the sockect and will start a new client thread on each connection with the socket and data reference.

12.7.2 Function Documentation

12.7.2.1 struct fwsocket∗ unixsocket_client (const char ∗ *sock,* int *protocol,* socketrecv *read,* void ∗ *data*)

Create a client thread on the socket.

It is not recomended to use SOCK_DGRAM as it requires a socket endpoint [inode] created this is done in /tmp using the basename of the socket and 6 random chars. this file is set to have no permissions as we only need the inode.

Parameters

sock	Path to UNIX socket
protocol	Either SOCK_STREAM or SOCK_DGRAM, SOCK_STREAM is recomended.
read	Call back to call when read is ready.
data	Reference to data to be returned in read callback.

Returns

Socket file descriptor

Definition at line 310 of file unixsock.c.

References fwsocket::addr, fwsocket::flags, make_socket(), objref(), objunref(), fwsocket::sock, SOCK_FLAG_UNIX, strlenzero(), and sockstruct::un.

Referenced by unixsocktest().

```
00310                                                                                                                {
00311     struct fwsocket *fws;
00312     union sockstruct caddr, *saddr;
00313     char *temp = NULL;
00314     const char *tmpsock;
00315     int salen;
00316     mode_t omask;
00317 
00318     /*Create a UNIX socket structure*/
00319     if (!(fws = make_socket(PF_UNIX, protocol, 0, NULL))) {
00320         return NULL;
00321     }
00322 
00323     /* bind my endpoint to temp file*/
00324     if (protocol == SOCK_DGRAM) {
00325         /*yip i want only a inode here folks*/
00326         omask = umask(S_IXUSR | S_IRUSR | S_IWUSR | S_IWGRP | S_IRGRP | S_IXGRP | S_IWOTH | S_IROTH |
      S_IXOTH);
00327         tmpsock = basename((char*)sock);
00328         temp = tempnam(NULL, tmpsock);
00329         if (strlenzero(temp)) {
00330             if (temp) {
00331                 free(temp);
00332             }
00333             objunref(fws);
00334             return NULL;
00335         }
00336 
00337         /*Allocate address and connect to the client*/
00338         salen = sizeof(caddr.un);
00339         memset(&caddr.un, 0, salen);
00340         caddr.un.sun_family = PF_UNIX;
00341         strncpy((char *)caddr.un.sun_path, temp, sizeof(caddr.un.sun_path) -1);
00342 
00343         if (bind(fws->sock, (struct sockaddr *)&caddr.un, salen)) {
00344             /*reset umask*/
00345             umask(omask);
00346             if (temp) {
00347                 if (!strlenzero(temp)) {
00348                     unlink(temp);
00349                 }
00350                 free(temp);
00351             }
00352             objunref(fws);
00353             return NULL;
00354         }
00355         /*reset umask*/
00356         umask(omask);
00357     }
00358 
00359     /*Allocate address and connect to the server*/
00360     saddr =  &fws->addr;
00361     salen = sizeof(saddr->un);
00362     memset(&saddr->un, 0, salen);
00363     saddr->un.sun_family = PF_UNIX;
00364     strncpy((char *)saddr->un.sun_path, sock, sizeof(saddr->un.sun_path) -1);
00365 
00366     if (connect(fws->sock, (struct sockaddr *)&saddr->un, salen)) {
00367         if (temp) {
00368             if (!strlenzero(temp)) {
00369                 unlink(temp);
00370             }
00371             free(temp);
00372         }
00373         objunref(fws);
00374         return NULL;
00375     }
00376 
00377     fws->flags |= SOCK_FLAG_UNIX;
00378     if (!(new_unixclientthread(fws, temp, read, data))) {
00379         if (temp) {
00380             if (!strlenzero(temp)) {
00381                 unlink(temp);
00382             }
00383             free(temp);
00384         }
00385         objunref(fws);
00386         return NULL;
00387     }
00388 
00389     return (objref(fws)) ? fws : NULL;
00390 }
```

12.7.2.2 struct fwsocket* unixsocket_server (const char * *sock,* int *protocol,* int *mask,* socketrecv *read,* void * *data*)

Create and run UNIX server socket thread.

Parameters

sock	Path to UNIX socket.
protocol	Protocol number.
mask	Umask for the socket.
read	Callback to call when there is data available.
data	Data reference to pass to read callback.

Returns

Reference to a socket

Definition at line 277 of file unixsock.c.

References unixserv_sockthread::data, framework_mkthread(), make_socket(), unixserv_sockthread::mask, objalloc(), objref(), objunref(), unixserv_sockthread::protocol, unixserv_sockthread::read, unixserv_sockthread::sock, and unixserv_sockthread::sockpath.

Referenced by unixsocktest().

```
00277
          {
00278     struct unixserv_sockthread *unsock;
00279
00280     if (!(unsock = objalloc(sizeof(*unsock), free_unixserv))) {
00281         return NULL;
00282     }
00283
00284     strncpy(unsock->sockpath, sock, UNIX_PATH_MAX);
00285     unsock->mask = mask;
00286     unsock->read = read;
00287     unsock->protocol = protocol;
00288     unsock->data = (objref(data)) ? data : NULL;
00289
00290     /*Create a UNIX socket structure*/
00291     if (!(unsock->sock = make_socket(PF_UNIX, protocol, 0, NULL))) {
00292         objunref(unsock);
00293         return NULL;
00294     }
00295
00296     framework_mkthread(unsock_serv, NULL, NULL, unsock, 0);
00297     return (objref(unsock->sock)) ? unsock->sock : NULL;
00298 }
```

12.8 Multicast sockets

Support for multicast sockets either IPv4 or IPv6.

Functions

- void mcast6_ip (struct in6_addr *addr)

  Randomally assign a SSM Multicast address.
  param addr Ip address structure to fill out.
- void mcast4_ip (struct in_addr *addr)

  Randomally assign a SSM Multicast address.
- struct fwsocket * mcast_socket (const char *iface, int family, const char *mcastip, const char *port, int flags)

  Create a multicast socket.

12.8.1 Detailed Description

Support for multicast sockets either IPv4 or IPv6.

12.8.2 Function Documentation

12.8.2.1 void mcast4_ip (struct in_addr * *addr*)

Randomally assign a SSM Multicast address.

Parameters

addr	Ip address structure to fill out.

Definition at line 504 of file iputil.c.

References genrand().

Referenced by mcast_socket().

```
00504                                       {
00505     uint32_t mip, rand;
00506
00507     do {
00508         rand = genrand(&mip, 3);
00509         mip >>= 8;
00510     } while (!rand || !(mip >> 8));
00511     mip |= 232 << 24;
00512
00513     addr->s_addr = htonl(mip);
00514 }
```

12.8.2.2 void mcast6_ip (struct in6_addr * *addr*)

Randomally assign a SSM Multicast address.

param addr Ip address structure to fill out.

Definition at line 480 of file iputil.c.

References genrand().

Referenced by mcast_socket().

```
00480                                       {
00481     int mip, rand;
00482     uint32_t *i;
00483
```

```
00484 #ifndef __WIN32
00485     i = (uint32_t*)&addr->s6_addr32;
00486 #else
00487     i = (uint32_t*)&addr->u.Word;
00488 #endif
00489     i[0] = htonl(0xFF350000);
00490     i[1] = 0;
00491     i[2] = 0;
00492     i[3] = 1 << 31;
00493 
00494     do {
00495         rand = genrand(&mip, 4);
00496     } while (!rand);
00497 
00498     i[3] = htonl(i[3] | mip);
00499 }
```

12.8.2.3 struct fwsocket* mcast_socket (const char * *iface*, int *family*, const char * *mcastip*, const char * *port*, int *flags*)

Create a multicast socket.

A multicast socket is both a client and server due to the nature of multicasting writing to a multicast socket should only be done with socketwrite not socketwrite_d the socket is created on a interface and the initial address can be set.

Todo Win32 support for inet_ntop/inet_pton

Parameters

iface	Interface to send and recieve multicast traffic.
family	IP address family PF_INET or PF_INET6.
mcastip	Multicast ip to use must be in "family".
port	Port to use.
flags	Multicast flags currently disables LOOP.

Returns

Reference to multicast ocket structure.

Definition at line 536 of file socket.c.

References fwsocket::addr, fwsocket::flags, get_iface_index(), get_ifinfo(), get_ifipaddr(), ifinfo::idx, inet_lookup(), ifinfo::ipv4addr, ifinfo::ipv6addr, make_socket(), mcast4_ip(), mcast6_ip(), objref(), objunref(), sockstruct::sa, seedrand(), fwsocket::sock, and SOCK_FLAG_MCAST.

```
00536 
      {
00537     struct fwsocket *fws;
00538     struct  addrinfo hint, *result, *rp;
00539     struct in_addr *srcif;
00540     const char *srcip;
00541     int ifidx;
00542     int on = 1;
00543     int off = 0;
00544     int ttl = 50;
00545     socklen_t slen = sizeof(union sockstruct);
00546 #ifdef __WIN32
00547     struct ifinfo *ifinf;
00548 #endif
00549 
00550         memset(&hint, 0, sizeof(hint));
00551     hint.ai_family = PF_UNSPEC;
00552     hint.ai_socktype = SOCK_DGRAM;
00553     hint.ai_protocol = IPPROTO_UDP;
00554 
00555 #ifndef __WIN32
00556     if (!(srcip = get_ifipaddr(iface, family))) {
00557                 return NULL;
00558     }
00559 
00560         if (getaddrinfo(srcip, port, &hint, &result) || !result) {
```

```
00561         free((void*)srcip);
00562                 return NULL;
00563         }
00564     free((void*)srcip);
00565 #else
00566     if (!(ifinf = get_ifinfo(iface))) {
00567         return NULL;
00568     }
00569     ifidx = ifinf->idx;
00570
00571     srcip = (family == AF_INET) ? ifinf->ipv4addr : ifinf->ipv6addr;
00572         if (!srcip || (getaddrinfo(srcip, port, &hint, &result) || !result)) {
00573         objunref(ifinf);
00574                 return NULL;
00575         }
00576     objunref(ifinf);
00577 #endif
00578
00579     for(rp = result; rp; rp = result->ai_next) {
00580         if (!(fws = make_socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol, NULL))) {
00581             continue;
00582         }
00583         break;
00584     }
00585
00586     if (!rp || !fws) {
00587         freeaddrinfo(result);
00588         return NULL;
00589     }
00590
00591     if(setsockopt(fws->sock, SOL_SOCKET, SO_REUSEADDR, (char*)&on, sizeof(on))) {
00592         objunref(fws);
00593         freeaddrinfo(result);
00594         return NULL;
00595     }
00596
00597     if (rp->ai_family == PF_INET) {
00598         struct in_addr mcastip4;
00599         struct ip_mreq mg;
00600         struct sockaddr_in *src_ip;
00601
00602         src_ip = (struct sockaddr_in*)rp->ai_addr;
00603
00604         if (setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_TTL, (char*)&ttl, sizeof(ttl))) {
00605             objunref(fws);
00606             freeaddrinfo(result);
00607             return NULL;
00608         }
00609
00610         if (flags && setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_LOOP, (char*)&off, sizeof(off))) {
00611             freeaddrinfo(result);
00612             objunref(fws);
00613             return NULL;
00614         }
00615
00616         if (mcastip) {
00617             inet_lookup(PF_INET, mcastip, &mcastip4, sizeof(mcastip4));
00618         } else {
00619             seedrand();
00620             mcast4_ip(&mcastip4);
00621         }
00622
00623         mg.imr_multiaddr = mcastip4;
00624         mg.imr_interface.s_addr = src_ip->sin_addr.s_addr;
00625         if (setsockopt(fws->sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char*)&mg, sizeof(mg))) {
00626             objunref(fws);
00627             freeaddrinfo(rp);
00628             return NULL;
00629         }
00630
00631         memset(&srcif, 0, sizeof(srcif));
00632         srcif = &src_ip->sin_addr;
00633         if(setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_IF, (char*)srcif, sizeof(*srcif))) {
00634             freeaddrinfo(rp);
00635             objunref(fws);
00636             return NULL;
00637         }
00638         src_ip->sin_addr.s_addr = mcastip4.s_addr;
00639     } else if (rp->ai_family == PF_INET6) {
00640         struct in6_addr mcastip6;
00641         struct ipv6_mreq mg;
00642         struct sockaddr_in6 *src_ip;
00643
00644 #ifndef __WIN32
00645         ifidx = get_iface_index(iface);
00646 #endif
00647         src_ip = (struct sockaddr_in6*)rp->ai_addr;
```

```
00648
00649         if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_HOPS, (char*)&ttl, sizeof(ttl))) {
00650             objunref(fws);
00651             freeaddrinfo(result);
00652             return NULL;
00653         }
00654
00655         if (flags && setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_LOOP, (char*)&off, sizeof(off))
      ) {
00656             freeaddrinfo(result);
00657             objunref(fws);
00658             return NULL;
00659         }
00660
00661         if (mcastip) {
00662             inet_lookup(PF_INET6, mcastip, &mcastip6, sizeof(mcastip6));
00663         } else {
00664             seedrand();
00665             mcast6_ip(&mcastip6);
00666         }
00667
00668         mg.ipv6mr_multiaddr = mcastip6;
00669         mg.ipv6mr_interface = ifidx;
00670         if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_JOIN_GROUP, (char*)&mg, sizeof(mg))) {
00671             objunref(fws);
00672             freeaddrinfo(rp);
00673             return NULL;
00674         }
00675
00676         if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_IF, (char*)&ifidx, sizeof(ifidx))) {
00677             objref(fws);
00678             freeaddrinfo(rp);
00679             return NULL;
00680         }
00681
00682         src_ip->sin6_addr = mcastip6;
00683     }
00684
00685     if (bind(fws->sock, (struct sockaddr*)rp->ai_addr, sizeof(struct sockaddr_storage))) {
00686         freeaddrinfo(result);
00687         objunref(fws);
00688         return NULL;
00689     }
00690
00691     getsockname(fws->sock, &fws->addr.sa, &slen);
00692     freeaddrinfo(result);
00693     fws->flags |= SOCK_FLAG_MCAST;
00694
00695     return fws;
00696 }
```

12.9 Linux network interface functions

Implement various interface routines from libnetlink.

Files

- file interface.c

  Wrapper arround Linux libnetlink for managing network interfaces.

Data Structures

- struct iplink_req

  IP Netlink request.
- struct ipaddr_req

  IP Netlink IP addr request.

Enumerations

- enum ipv4_score { IPV4_SCORE_ZEROCONF = 1 << 0, IPV4_SCORE_RESERVED = 1 << 1, IPV4_SCORE_ROUTABLE = 1 << 2 }

  Order of precidence of ipv4.
- enum ipv6_score { IPV6_SCORE_RESERVED = 1 << 0, IPV6_SCORE_SIXIN4 = 1 << 1, IPV6_SCORE_ROUTABLE = 1 << 2 }

  Return best ipv6 address in order of FFC/7 2002/16 ...

Functions

- void closenetlink ()

  Close netlink socket on application termination.
- int get_iface_index (const char *ifname)

  Get the netlink interface for a named interface.
- int delete_kernvlan (char *ifname, int vid)

  Delete a VLAN.
- int create_kernvlan (char *ifname, unsigned short vid)

  Create a VLAN on a interface.
- int delete_kernmac (char *ifname)

  Delete Kernel MAC VLAN.
- int create_kernmac (char *ifname, char *macdev, unsigned char *mac)

  Create a kernal MAC VLAN.
- int set_interface_flags (int ifindex, int set, int clear)

  Alter interface flags.
- int set_interface_addr (int ifindex, const unsigned char *hwaddr)

  Set interface MAC addr.
- int set_interface_name (int ifindex, const char *name)

  Rename interface.
- int interface_bind (char *iface, int protocol)

  Bind to device fd may be a existing socket.
- void randhwaddr (unsigned char *addr)

  create random MAC address
- int create_tun (const char *ifname, const unsigned char *hwaddr, int flags)

Create a tunnel device.

- int ifdown (const char *ifname, int flags)

  Set interface down.
- int ifup (const char *ifname, int flags)

  Set interface up.
- int ifrename (const char *oldname, const char *newname)

  Rename interface helper.
- int ifhwaddr (const char *ifname, unsigned char *hwaddr)

  Get MAC addr for interface.
- int set_interface_ipaddr (char *ifname, char *ipaddr)

  Set IP addr on interface.
- const char * get_ifipaddr (const char *iface, int family)

  Find best IP adress for a interface.

12.9.1 Detailed Description

Implement various interface routines from libnetlink.

12.9.2 Enumeration Type Documentation

12.9.2.1 enum ipv4_score

Order of precidence of ipv4.

Enumerator

IPV4_SCORE_ZEROCONF Zeroconf IP's 169.254/16.

IPV4_SCORE_RESERVED Reseverd "private" ip addresses.

IPV4_SCORE_ROUTABLE Routable IP's.

Definition at line 63 of file interface.c.

```
00063                 {
00065     IPV4_SCORE_ZEROCONF = 1 << 0,
00067     IPV4_SCORE_RESERVED = 1 << 1,
00069     IPV4_SCORE_ROUTABLE = 1 << 2
00070 };
```

12.9.2.2 enum ipv6_score

Return best ipv6 address in order of FFC/7 2002/16 ...

Enumerator

IPV6_SCORE_RESERVED Adminstrivly allocated addresses (FC/7)

IPV6_SCORE_SIXIN4 6in4 address space

IPV6_SCORE_ROUTABLE Other routable addresses.

Definition at line 73 of file interface.c.

```
00073                 {
00075     IPV6_SCORE_RESERVED = 1 << 0,
00077     IPV6_SCORE_SIXIN4 = 1 << 1,
00079     IPV6_SCORE_ROUTABLE = 1 << 2
00080 };
```

12.9.3 Function Documentation

12.9.3.1 void closenetlink (void)

Close netlink socket on application termination.

Definition at line 130 of file interface.c.

References objunref().

```
00130                               {
00131     if (nlh) {
00132         objunref(nlh);
00133     }
00134 }
```

12.9.3.2 int create_kernmac (char * *ifname*, char * *macdev*, unsigned char * *mac*)

Create a kernal MAC VLAN.

Parameters

ifname	Interface name to create
macdev	Base interface
mac	MAC address to use or random if NULL.

Returns

-1 on error.

Definition at line 282 of file interface.c.

References get_iface_index(), iplink_req::n, objalloc(), objlock(), objref(), objunlock(), objunref(), randhwaddr(), and strlenzero().

```
00282                                                                       {
00283     struct iplink_req *req;
00284     struct rtattr *data, *linkinfo;
00285     unsigned char lmac[ETH_ALEN];
00286     char *type = "macvlan";
00287     int ifindex, ret;
00288
00289     if (strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
00290             strlenzero(macdev) || (strlen(macdev) > IFNAMSIZ) ||
00291             (!objref(nlh) && !(nlh = nlhandle(0)))) {
00292         return (-1);
00293     }
00294
00295     /*set the index of base interface*/
00296     if (!(ifindex = get_iface_index(ifname))) {
00297         objunref(nlh);
00298         return (-1);
00299     }
00300
00301     if (!mac) {
00302         randhwaddr(lmac);
00303     } else {
00304         strncpy((char *)lmac, (char *)mac, ETH_ALEN);
00305     }
00306
00307     if (!(req = objalloc(sizeof(*req), NULL))) {
00308         objunref(nlh);
00309         return (-1);
00310     }
00311
00312     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00313     req->n.nlmsg_type = RTM_NEWLINK;
00314     req->n.nlmsg_flags = NLM_F_CREATE | NLM_F_EXCL | NLM_F_REQUEST;
00315
00316     /*config base/dev/mac*/
00317     addattr_l(&req->n, sizeof(*req), IFLA_LINK, &ifindex, 4);
00318     addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, macdev, strlen(macdev));
00319     addattr_l(&req->n, sizeof(*req), IFLA_ADDRESS, lmac, ETH_ALEN);
```

```
00320
00321     /*type*/
00322     linkinfo  = NLMSG_TAIL(&req->n);
00323     addattr_l(&req->n, sizeof(*req), IFLA_LINKINFO, NULL, 0);
00324     addattr_l(&req->n, sizeof(*req), IFLA_INFO_KIND, type, strlen(type));
00325
00326     /*mode*/
00327     data = NLMSG_TAIL(&req->n);
00328     addattr_l(&req->n, sizeof(*req), IFLA_INFO_DATA, NULL, 0);
00329     addattr32(&req->n, sizeof(*req), IFLA_MACVLAN_MODE, MACVLAN_MODE_PRIVATE);
00330     data->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)data;
00331     linkinfo->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)linkinfo;
00332
00333     objlock(nlh);
00334     ret = rtnl_talk(nlh, &req->n, 0, 0, NULL);
00335     objunlock(nlh);
00336
00337     objunref(nlh);
00338     objunref(req);
00339
00340     return (ret);
00341 }
```

12.9.3.3 int create_kernvlan (char * *ifname*, unsigned short *vid*)

Create a VLAN on a interface.

Parameters

ifname	Interface to add VLAN to.
vid	VLAN id to add.

Returns

-1 on error.

Definition at line 214 of file interface.c.

References get_iface_index(), iplink_req::n, objalloc(), objlock(), objref(), objunlock(), objunref(), and strlenzero().

```
00214                                                          {
00215     struct iplink_req *req;
00216     char iface[IFNAMSIZ+1];
00217     struct rtattr *data, *linkinfo;
00218     char *type = "vlan";
00219     int ifindex, ret;
00220
00221     if (strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
00222             (!objref(nlh) && !(nlh = nlhandle(0)))) {
00223         return (-1);
00224     }
00225
00226     /*set the index of base interface*/
00227     if (!(ifindex = get_iface_index(ifname))) {
00228         objunref(nlh);
00229         return (-1);
00230     }
00231
00232     if (!(req = objalloc(sizeof(*req), NULL))) {
00233         objunref(nlh);
00234         return (-1);
00235     }
00236
00237     snprintf(iface, IFNAMSIZ, "%s.%i", ifname, vid);
00238     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00239     req->n.nlmsg_type = RTM_NEWLINK;
00240     req->n.nlmsg_flags = NLM_F_CREATE | NLM_F_EXCL | NLM_F_REQUEST;
00241
00242     /*config base/dev/mac*/
00243     addattr_l(&req->n, sizeof(*req), IFLA_LINK, &ifindex, sizeof(ifindex));
00244     addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, iface, strlen(iface));
00245
00246     /*type*/
00247     linkinfo  = NLMSG_TAIL(&req->n);
00248     addattr_l(&req->n, sizeof(*req), IFLA_LINKINFO, NULL, 0);
00249     addattr_l(&req->n, sizeof(*req), IFLA_INFO_KIND, type, strlen(type));
00250
00251     /*vid*/
```

```
00252     data = NLMSG_TAIL(&req->n);
00253     addattr_l(&req->n, sizeof(*req), IFLA_INFO_DATA, NULL, 0);
00254     addattr_l(&req->n, sizeof(*req), IFLA_VLAN_ID, &vid, sizeof(vid));
00255
00256     data->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)data;
00257     linkinfo->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)linkinfo;
00258
00259     objlock(nlh);
00260     ret = rtnl_talk(nlh, &req->n, 0, 0, NULL);
00261     objunlock(nlh);
00262
00263     objunref(nlh);
00264     objunref(req);
00265
00266     return (ret);
00267 }
```

12.9.3.4 int create_tun (const char * *ifname*, const unsigned char * *hwaddr*, int *flags*)

Create a tunnel device.

Parameters

ifname	Interface name to create..
hwaddr	Hardware address to assign (optionally).
flags	Flags to set device properties.

Returns

Tunnel FD or -1 on error.

Definition at line 496 of file interface.c.

References get_iface_index(), set_interface_addr(), and set_interface_flags().

```
00496                                                                                {
00497     struct ifreq ifr;
00498     int fd, ifindex;
00499     char *tundev = "/dev/net/tun";
00500
00501     /* open the tun/tap clone dev*/
00502     if ((fd = open(tundev, O_RDWR)) < 0) {
00503         return (-1);
00504     }
00505
00506     /* configure the device*/
00507     memset(&ifr, 0, sizeof(ifr));
00508     ifr.ifr_flags = flags;
00509     strncpy(ifr.ifr_name, ifname, IFNAMSIZ);
00510     if (ioctl(fd, TUNSETIFF, (void *)&ifr) < 0 ) {
00511         perror("ioctl(TUNSETIFF) failed\n");
00512         close(fd);
00513         return (-1);
00514     }
00515
00516     if (!(ifindex = get_iface_index(ifname))) {
00517         return (-1);
00518     }
00519
00520     /* set the MAC address*/
00521     if (hwaddr) {
00522         set_interface_addr(ifindex, hwaddr);
00523     }
00524
00525     /*set the network dev up*/
00526     set_interface_flags(ifindex, IFF_UP | IFF_RUNNING | IFF_MULTICAST | IFF_BROADCAST, 0
      );
00527
00528     return (fd);
00529 }
```

12.9.3.5 int delete_kernmac (char * *ifname*)

Delete Kernel MAC VLAN.

Parameters

ifname	Interface to delete.

Returns

-1 on error.

Definition at line 272 of file interface.c.

```
00272                                       {
00273
00274     return (delete_interface(ifname));
00275 }
```

12.9.3.6 int delete_kernvlan (char * *ifname*, int *vid*)

Delete a VLAN.

Parameters

ifname	Interface we deleting vlan from.
vid	VLAN id to delete.

Returns

-1 on error.

Definition at line 201 of file interface.c.

```
00201                                                 {
00202     char iface[IFNAMSIZ+1];
00203
00204     /*check ifname grab a ref to nlh or open it*/
00205     snprintf(iface, IFNAMSIZ, "%s.%i", ifname, vid);
00206     return (delete_interface(iface));
00207 }
```

12.9.3.7 int get_iface_index (const char * *ifname*)

Get the netlink interface for a named interface.

Parameters

ifname	Interface name.

Returns

Index of the interface.

Definition at line 139 of file interface.c.

References objlock(), objref(), objunlock(), and objunref().

Referenced by create_kernmac(), create_kernvlan(), create_tun(), ifdown(), ifhwaddr(), ifrename(), ifup(), interface_bind(), mcast_socket(), and set_interface_ipaddr().

```
00139                                               {
00140     int ifindex;
00141
00142     if (!objref(nlh) && !(nlh = nlhandle(0))) {
00143         return (0);
00144     }
```

```
00145
00146     objlock(nlh);
00147     ll_init_map(nlh, 1);
00148     objunlock(nlh);
00149
00150     ifindex = ll_name_to_index(ifname);
00151
00152     objunref(nlh);
00153     return (ifindex);
00154 }
```

12.9.3.8 const char* get_ifipaddr (const char * *iface*, int *family*)

Find best IP adress for a interface.

Todo WIN32 Support

Parameters

iface	Interface name.
family	PF_INET or PF_INET6.

Returns

Best matching IP address for the interface.

Definition at line 783 of file interface.c.

References score_ipv4(), score_ipv6(), and strlenzero().

Referenced by mcast_socket().

```
00783                                                                   {
00784     struct ifaddrs *ifaddr, *ifa;
00785     struct sockaddr_in *ipv4addr;
00786     int score = 0, nscore, iflen;
00787     uint32_t subnet = 0, match;
00788     char host[NI_MAXHOST] = "", tmp[NI_MAXHOST];
00789
00790     if (!iface || getifaddrs(&ifaddr) == -1) {
00791         return NULL;
00792     }
00793
00794     for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
00795         iflen = strlen(iface);
00796         if ((ifa->ifa_addr == NULL) ||  strncmp(ifa->ifa_name, iface, iflen) || (ifa->ifa_addr->sa_family
      != family)) {
00797             continue;
00798         }
00799
00800         /* Match aliases not vlans*/
00801         if ((strlen(ifa->ifa_name) > iflen) && (ifa->ifa_name[iflen] != ':')) {
00802             continue;
00803         }
00804
00805         switch (ifa->ifa_addr->sa_family) {
00806             case AF_INET:
00807                 /* Find best ip address for a interface lowest priority is given to zeroconf then reserved
       ip's
00808                  * finally find hte ip with shortest subnet bits.*/
00809                 ipv4addr = (struct sockaddr_in*)ifa->ifa_netmask;
00810                 match = ntohl(~ipv4addr->sin_addr.s_addr);
00811
00812                 nscore = score_ipv4((struct sockaddr_in*)ifa->ifa_addr, tmp, NI_MAXHOST);
00813
00814                 /* match score and subnet*/
00815                 if  ((nscore > score) || ((nscore == score) && (match > subnet))) {
00816                     score = nscore;
00817                     subnet = match;
00818                     strncpy(host, tmp, NI_MAXHOST);
00819                 }
00820                 break;
00821             case AF_INET6:
00822                 nscore = score_ipv6((struct sockaddr_in6*)ifa->ifa_addr, tmp, NI_MAXHOST);
```

```
00823
00824                 if (nscore > score) {
00825                     score = nscore;
00826                     strncpy(host, tmp, NI_MAXHOST);
00827                 }
00828                 break;
00829             }
00830     }
00831     freeifaddrs(ifaddr);
00832     return (strlenzero(host)) ? NULL : strdup(host);
00833 }
```

12.9.3.9 int ifdown (const char * *ifname*, int *flags*)

Set interface down.

Parameters

ifname	Interface name.
flags	Additional flags to clear.

Returns

-1 on error 0 on success.

Definition at line 535 of file interface.c.

References get_iface_index(), and set_interface_flags().

Referenced by ifrename().

```
00535                                                 {
00536     int ifindex;
00537
00538     /*down the device*/
00539     if (!(ifindex = get_iface_index(ifname))) {
00540         return (-1);
00541     }
00542
00543     /*set the network dev up*/
00544     set_interface_flags(ifindex, 0, IFF_UP | IFF_RUNNING | flags);
00545
00546     return (0);
00547 }
```

12.9.3.10 int ifhwaddr (const char * *ifname*, unsigned char * *hwaddr*)

Get MAC addr for interface.

Parameters

ifname	Interface name
hwaddr	Buffer to place MAC in char[ETH_ALEN]

Returns

0 on success.

Definition at line 588 of file interface.c.

References get_iface_index(), objref(), objunref(), and strlenzero().

Referenced by get_ip6_addrprefix().

```
00588                                                                 {
00589     int ifindex;
00590
```

```
00591     if (!hwaddr || strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
00592             (!objref(nlh) && !(nlh = nlhandle(0)))) {
00593         return (-1);
00594     }
00595
00596     /*set the index of base interface*/
00597     if (!(ifindex = get_iface_index(ifname))) {
00598         objunref(nlh);
00599         return (-1);
00600     }
00601
00602     ll_index_to_addr(ifindex, hwaddr, ETH_ALEN);
00603     objunref(nlh);
00604     return (0);
00605 }
```

12.9.3.11 int ifrename (const char * *oldname*, const char * *newname*)

Rename interface helper.

Parameters

oldname	Original name.
newname	New name.

Returns

0 on success.

Definition at line 571 of file interface.c.

References get_iface_index(), ifdown(), and set_interface_name().

```
00571                                                           {
00572     int ifindex;
00573
00574     ifdown(oldname, 0);
00575
00576     if (!(ifindex = get_iface_index(oldname))) {
00577         return (-1);
00578     }
00579     set_interface_name(ifindex, newname);
00580
00581     return (0);
00582 }
```

12.9.3.12 int ifup (const char * *ifname*, int *flags*)

Set interface up.

Parameters

ifname	Interface name.
flags	Additional flags to set.

Returns

-1 on error 0 on success.

Definition at line 553 of file interface.c.

References get_iface_index(), and set_interface_flags().

```
00553                                           {
00554     int ifindex;
00555
00556     /*down the device*/
```

```
00557     if (!(ifindex = get_iface_index(ifname))) {
00558         return (-1);
00559     }
00560
00561     /*set the network dev up*/
00562     set_interface_flags(ifindex, IFF_UP | IFF_RUNNING | flags, 0);
00563
00564     return (0);
00565 }
```

12.9.3.13 int interface_bind (char * *iface*, int *protocol*)

Bind to device fd may be a existing socket.

Parameters

iface	Interface to bind too.
protocol	Protocol to use.

Returns

-1 on error.

Definition at line 453 of file interface.c.

References get_iface_index(), and set_interface_flags().

```
00453                                                   {
00454     struct sockaddr_ll sll;
00455     int proto = htons(protocol);
00456     int fd, ifindex;
00457
00458     /*set the network dev up*/
00459     if (!(ifindex = get_iface_index(iface))) {
00460         return (-1);
00461     }
00462     set_interface_flags(ifindex, IFF_UP | IFF_RUNNING, 0);
00463
00464     /* open network raw socket */
00465     if ((fd = socket(PF_PACKET, SOCK_RAW,  proto)) < 0) {
00466         return (-1);
00467     }
00468
00469     /*bind to the interface*/
00470     memset(&sll, 0, sizeof(sll));
00471     sll.sll_family = PF_PACKET;
00472     sll.sll_protocol = proto;
00473     sll.sll_ifindex = ifindex;
00474     if (bind(fd, (struct sockaddr *)&sll, sizeof(sll)) < 0) {
00475         perror("bind failed");
00476         close(fd);
00477         return (-1);
00478     }
00479
00480     return (fd);
00481 }
```

12.9.3.14 void randhwaddr (unsigned char * *addr*)

create random MAC address

Parameters

addr	Buffer char[ETH_ALEN] filled with the new address.

Definition at line 485 of file interface.c.

References genrand().

Referenced by create_kernmac().

```
00485                                                {
00486     genrand(addr, ETH_ALEN);
00487     addr [0] &= 0xfe;       /* clear multicast bit */
00488     addr [0] |= 0x02;       /* set local assignment bit (IEEE802) */
00489 }
```

12.9.3.15 int set_interface_addr (int *ifindex*, const unsigned char * *hwaddr*)

Set interface MAC addr.

Parameters

ifindex	Interface index.
hwaddr	MAC address to set.

Returns

-1 on error.

Definition at line 388 of file interface.c.

References iplink_req::i, iplink_req::n, objalloc(), objlock(), objref(), objunlock(), and objunref().

Referenced by create_tun().

```
00388                                                                    {
00389     struct iplink_req *req;
00390
00391     if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
00392         return (-1);
00393     }
00394
00395     if (!(req = objalloc(sizeof(*req), NULL))) {
00396         objunref(nlh);
00397         return (-1);
00398     }
00399
00400     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00401     req->n.nlmsg_type = RTM_NEWLINK;
00402     req->n.nlmsg_flags = NLM_F_REQUEST;
00403     req->i.ifi_index = ifindex;
00404
00405     /*config base/dev/mac*/
00406     addattr_l(&req->n, sizeof(*req), IFLA_ADDRESS, hwaddr, ETH_ALEN);
00407
00408     objlock(nlh);
00409     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00410     objunlock(nlh);
00411
00412     objunref(nlh);
00413     objunref(req);
00414     return (0);
00415 }
```

12.9.3.16 int set_interface_flags (int *ifindex*, int *set*, int *clear*)

Alter interface flags.

Parameters

ifindex	Interface index.
set	Flags to set.
clear	Flags to clear.

Returns

-1 on error.

Definition at line 348 of file interface.c.

References iplink_req::i, iplink_req::n, objalloc(), objlock(), objref(), objunlock(), and objunref().

Referenced by create_tun(), ifdown(), ifup(), and interface_bind().

```
00348                                                                   {
00349     struct iplink_req *req;
00350     int flags;
00351
00352     if (!objref(nlh) && !(nlh = nlhandle(0))) {
00353         return (-1);
00354     }
00355
00356     flags = ll_index_to_flags(ifindex);
00357
00358     flags |= set;
00359     flags &= ~(clear);
00360
00361     if (!(req = objalloc(sizeof(*req), NULL))) {
00362         objunref(nlh);
00363         return (-1);
00364     }
00365
00366     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00367     req->n.nlmsg_type = RTM_NEWLINK;
00368     req->n.nlmsg_flags = NLM_F_REQUEST;
00369
00370     /*config base/dev/mac*/
00371     req->i.ifi_index = ifindex;
00372     req->i.ifi_flags = flags;
00373     req->i.ifi_change = set | clear;
00374
00375     objlock(nlh);
00376     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00377     objunlock(nlh);
00378
00379     objunref(nlh);
00380     objunref(req);
00381     return (0);
00382 }
```

12.9.3.17 int set_interface_ipaddr (char * *ifname*, char * *ipaddr*)

Set IP addr on interface.

Parameters

ifname	Interface to assign IP to
ipaddr	IP Addr to assign.

Returns

-1 on error.

Definition at line 611 of file interface.c.

References get_iface_index(), ipaddr_req::i, ipaddr_req::n, objalloc(), objlock(), objref(), objunlock(), and objunref().

```
00611                                                                   {
00612     struct ipaddr_req *req;
00613     inet_prefix lcl;
00614     int ifindex, bcast;
00615
00616     if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
00617         return (-1);
00618     }
00619
00620     if (!(req = objalloc(sizeof(*req), NULL))) {
00621         objunref(nlh);
00622         return (-1);
```

```
00623     }
00624
00625     /*set the index of base interface*/
00626     if (!(ifindex = get_iface_index(ifname))) {
00627         objunref(nlh);
00628         return (-1);
00629     }
00630
00631     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifaddrmsg));
00632     req->n.nlmsg_type = RTM_NEWADDR;
00633     req->n.nlmsg_flags = NLM_F_REQUEST | NLM_F_EXCL | NLM_F_CREATE;
00634
00635     req->i.ifa_scope = RT_SCOPE_HOST;
00636     req->i.ifa_index = ifindex;
00637
00638     get_prefix(&lcl, ipaddr, AF_UNSPEC);
00639     req->i.ifa_family = lcl.family;
00640     req->i.ifa_prefixlen = lcl.bitlen;
00641
00642     addattr_l(&req->n, sizeof(*req), IFA_LOCAL, &lcl.data, lcl.bytelen);
00643     addattr_l(&req->n, sizeof(*req), IFA_ADDRESS, &lcl.data, lcl.bytelen);
00644     if (lcl.family == AF_INET) {
00645         bcast = htonl((1 << (32 - lcl.bitlen)) - 1);
00646         addattr32(&req->n, sizeof(*req), IFA_BROADCAST, lcl.data[0] | bcast);
00647     }
00648
00649     objlock(nlh);
00650     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00651     objunlock(nlh);
00652
00653     objunref(nlh);
00654     objunref(req);
00655     return (0);
00656 }
```

12.9.3.18 int set_interface_name (int *ifindex*, const char * *name*)

Rename interface.

Parameters

ifindex	Interface index.
name	New interface name.

Returns

-1 on error.

Definition at line 421 of file interface.c.

References iplink_req::i, iplink_req::n, objalloc(), objlock(), objref(), objunlock(), and objunref().

Referenced by ifrename().

```
00421                                                   {
00422     struct iplink_req *req;
00423
00424     if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
00425         return (-1);
00426     }
00427
00428     if (!(req = objalloc(sizeof(*req), NULL))) {
00429         objunref(nlh);
00430         return (-1);
00431     }
00432
00433     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00434     req->n.nlmsg_type = RTM_NEWLINK;
00435     req->n.nlmsg_flags = NLM_F_REQUEST;
00436     req->i.ifi_index = ifindex;
00437
00438     addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, name, strlen((char *)name));
00439
00440     objlock(nlh);
00441     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00442     objunlock(nlh);
00443
```

```
00444     objunref(nlh);
00445     objunref(req);
00446     return (0);
00447 }
```

12.10 INI Style config file Interface

Reads a ini config file into grouped hashed buckets.

Files

- file config.c

  INI style config file interface.

Data Structures

- struct config_category

  Configuration file category.
- struct config_file

  Config file.
- struct config_entry

  Configuration category entry.

Typedefs

- typedef void(∗ config_filecb)(struct bucket_list ∗, const char ∗, const char ∗)

  Calback used when processing config files.
- typedef void(∗ config_catcb)(struct bucket_list ∗, const char ∗)

  Calback used when processing a category.
- typedef void(∗ config_entrycb)(const char ∗, const char ∗)

  Callback used when processing a entry.

Functions

- void unrefconfigfiles (void)

  Empty out and unreference config files.
- int process_config (const char ∗configname, const char ∗configfile)

  Process a configfile into buckets.
- struct bucket_list ∗ get_config_file (const char ∗configname)

  Returns the catergories bucket for a config file.
- struct bucket_list ∗ get_config_category (const char ∗configname, const char ∗category)

  Return a single category.
- struct bucket_list ∗ get_category_next (struct bucket_loop ∗cloop, char ∗name, int len)

  Iterate through categories returning the entries bucket.
- struct bucket_loop ∗ get_category_loop (const char ∗configname)

  Return a bucket loop to allow iterating over categories.
- void config_entry_callback (struct bucket_list ∗entries, config_entrycb entry_cb)

  Callback Wraper that iterates through all items calling a callback for each item.
- void config_cat_callback (struct bucket_list ∗categories, config_catcb cat_cb)

  Callback wrapper that iterates through categories calling a callback on each category.
- void config_file_callback (config_filecb file_cb)

  Callback wrapper to iterate over all configfiles calling a callback on each file.
- struct config_entry ∗ get_config_entry (struct bucket_list ∗categories, const char ∗item)

  Find the entry in a config file.

12.10.1 Detailed Description

Reads a ini config file into grouped hashed buckets.

12.10.2 Typedef Documentation

12.10.2.1 typedef void(* config_catcb)(struct bucket_list *, const char *)

Calback used when processing a category.

Parameters

entries	Bucket list containing entries.
name	Category name.

Definition at line 291 of file dtsapp.h.

12.10.2.2 typedef void(* config_entrycb)(const char *, const char *)

Callback used when processing a entry.

Parameters

item	Name of the entry.
value	Value of the entry.

Definition at line 297 of file dtsapp.h.

12.10.2.3 typedef void(* config_filecb)(struct bucket_list *, const char *, const char *)

Calback used when processing config files.

Parameters

categories	Bucket list of categories.
filename	The filename.
filepath	The filepath.

Definition at line 285 of file dtsapp.h.

12.10.3 Function Documentation

12.10.3.1 void config_cat_callback (struct bucket_list * *categories*, config_catcb *cat_cb*)

Callback wrapper that iterates through categories calling a callback on each category.

See Also

config_catcb

Parameters

categories	Bucketlist from a config file containing categories.
cat_cb	Callback to call on each category.

Definition at line 383 of file config.c.

References bucketlist_callback().

```
00383                                                                                {
00384     bucketlist_callback(categories, category_callback, &cat_cb);
00385 }
```

12.10.3.2 void config_entry_callback (struct bucket_list * *entries*, config_entrycb *entry_cb*)

Callback Wraper that iterates through all items calling a callback for each item.

See Also

config_entrycb

Parameters

entries	Bucketlist of entries (from a category).
entry_cb	The callback to call on each entry.

Definition at line 365 of file config.c.

References bucketlist_callback().

```
00365                                                                                    {
00366     bucketlist_callback(entries, entry_callback, &entry_cb);
00367 }
```

12.10.3.3 void config_file_callback (config_filecb *file_cb*)

Callback wrapper to iterate over all configfiles calling a callback on each file.

See Also

config_filecb

Parameters

file_cb	Callback to call.

Definition at line 400 of file config.c.

References bucketlist_callback().

```
00400                                                {
00401     bucketlist_callback(configfiles, file_callback, &file_cb);
00402 }
```

12.10.3.4 struct bucket_loop* get_category_loop (const char * *confignam*e)

Return a bucket loop to allow iterating over categories.

Parameters

*confignam*e	Name assigned to the config file when calling process_config().

Returns

Bucket loop iterator.

Definition at line 341 of file config.c.

References get_config_file(), init_bucket_loop(), and objunref().

```
00341                                                                   {
00342     struct bucket_loop *cloop;
00343     struct bucket_list *file;
00344
00345     file = get_config_file(configname);
00346     cloop = init_bucket_loop(file);
00347     objunref(file);
00348     return (cloop);
00349 }
```

12.10.3.5 struct bucket_list* get_category_next (struct bucket_loop * *cloop*, char * *name*, int *len*)

Iterate through categories returning the entries bucket.

As well as the entries returned name will be filled upto len bytes with the category name

Parameters

cloop	Iterator created with get_category_loop.
name	Buffer where the category name is copied.
len	limit the number of characters copied to len.

Returns

Entries list for category returned in paramaater name.

Definition at line 317 of file config.c.

References config_category::entries, config_category::name, next_bucket_loop(), objref(), objunref(), and strlenzero().

```
00317                                                                                   {
00318     struct config_category *category;
00319
00320     if (cloop && (category = next_bucket_loop(cloop))) {
00321         if (category->entries) {
00322             if (!objref(category->entries)) {
00323                 objunref(category);
00324                 return (NULL);
00325             }
00326             if (!strlenzero(name)) {
00327                 strncpy(name, category->name, len);
00328             }
00329             objunref(category);
00330             return (category->entries);
00331         } else {
00332             objunref(category);
00333         }
00334     }
00335     return (NULL);
00336 }
```

12.10.3.6 struct bucket_list* get_config_category (const char * *configname*, const char * *category*)

Return a single category.

If category is NULL the category "default" is returned.

Parameters

configname	Name assigned to the config file when calling process_config().
category	Configuration category to return or "default" if NULL.

Returns

Bucket list containing the category.

Definition at line 286 of file config.c.

References bucket_list_find_key(), config_category::entries, get_config_file(), objref(), and objunref().

```
00286                                                                                {
00287     struct bucket_list *file;
00288     struct config_category *cat;
00289
00290     file = get_config_file(configname);
00291     if (category) {
00292         cat = bucket_list_find_key(file, category);
00293     } else {
00294         cat = bucket_list_find_key(file, "default");
00295     }
00296
00297     objunref(file);
00298     if (cat) {
00299         if (!objref(cat->entries)) {
00300             objunref(cat);
00301             return (NULL);
00302         }
00303         objunref(cat);
00304         return (cat->entries);
00305     } else {
00306         return (NULL);
00307     }
00308 }
```

12.10.3.7 struct config_entry* get_config_entry (struct bucket_list * *categories*, const char * *item*)

Find the entry in a config file.

Parameters

categories	Categories bucketlist.
item	Item to search for.

Returns

Reference to a entry.

Definition at line 408 of file config.c.

References bucket_list_find_key().

```
00408                                                                                {
00409     struct config_entry *entry;
00410
00411     entry = bucket_list_find_key(categories, item);
00412
00413     return (entry);
00414 }
```

12.10.3.8 struct bucket_list* get_config_file (const char * *configname*)

Returns the catergories bucket for a config file.

Parameters

configname	Name assigned to the config file when calling process_config().

Returns

Categories bucketlist.

Definition at line 263 of file config.c.

References bucket_list_find_key(), config_file::cat, objref(), and objunref().

Referenced by get_category_loop(), and get_config_category().

```
00263                                                                   {
00264     struct config_file *file;
00265
00266     if ((file = bucket_list_find_key(configfiles, configname))) {
00267         if (file->cat) {
00268             if (!objref(file->cat)) {
00269                 objunref(file);
00270                 return (NULL);
00271             }
00272             objunref(file);
00273             return (file->cat);
00274         }
00275         objunref(file);
00276     }
00277     return (NULL);
00278 }
```

12.10.3.9 int process_config (const char * *configname*, const char * *configfile*)

Process a configfile into buckets.

Parameters

configname	Name of the configuration.
configfile	File to load into this configuration container.

Returns

Zero on success.

Definition at line 197 of file config.c.

References addtobucket(), config_file::cat, config_file::filepath, objunref(), strlenzero(), and trim().

```
00197                                                                   {
00198     struct config_file *file;
00199     struct config_category *category = NULL;
00200     FILE *config;
00201     char line[256];
00202     char item[128];
00203     char value[128];
00204     char *tmp = (char *)&line;
00205     char *token;
00206
00207     if (!configfiles) {
00208         initconfigfiles();
00209     }
00210
00211     file = create_conf_file(configname, configfile);
00212     addtobucket(configfiles, file);
00213
00214     if (!(config = fopen(file->filepath, "r"))) {
00215         return (-1);
00216     }
00217
00218     while(fgets(line, sizeof(line) - 1, config)) {
00219         if (!(tmp = filterconf(line, 3))) {
00220             continue;
00221         }
00222
```

```
00223         /*this is a new category*/
00224         if ((token = strchr(tmp, '[')) && (token == tmp)) {
00225             tmp++;
00226             token = strrchr(tmp, ']');
00227             token[0] = '\0';
00228             tmp = trim(tmp);
00229             if (!strlenzero(tmp)) {
00230                 if (category) {
00231                     objunref(category);
00232                 }
00233                 category = create_conf_category(tmp);
00234                 addtobucket(file->cat, category);
00235             }
00236             continue;
00237         }
00238 
00239         if (sscanf(tmp, "%[^=] %*[=] %[^\n]", (char *)&item, (char *)&value) != 2) {
00240             continue;
00241         }
00242 
00243         if (!category) {
00244             category = create_conf_category("default");
00245             addtobucket(file->cat, category);
00246         }
00247 
00248         add_conf_entry(category, trim(item), trim(value));
00249     }
00250     fclose(config);
00251     if (category) {
00252         objunref(category);
00253     }
00254     if (file) {
00255         objunref(file);
00256     }
00257     return (0);
00258 }
```

12.10.3.10 void unrefconfigfiles (void)

Empty out and unreference config files.

Definition at line 78 of file config.c.

References objunref().

Referenced by framework_init().

```
00078                                   {
00079     if (configfiles) {
00080         objunref(configfiles);
00081     }
00082 }
```

12.11 Radius client interface

Simple implementation of experimental radius client.

Files

- file radius.c

  Simple radius client implementation.

Data Structures

- struct radius_packet

  Radius Packet.
- struct radius_session

  Radius session.
- struct radius_connection

  Radius connection.
- struct radius_server

  Radius Server.

Macros

- #define RAD_AUTH_HDR_LEN 20

  Authentification header length.
- #define RAD_AUTH_PACKET_LEN 4096

  Auth packet length.
- #define RAD_AUTH_TOKEN_LEN 16

  Auth token length.
- #define RAD_MAX_PASS_LEN 128

  Auth max password length.
- #define RAD_ATTR_USER_NAME 1 /∗string∗/

  Radius attribute username.
- #define RAD_ATTR_USER_PASSWORD 2 /∗passwd∗/

  Radius attribute password.
- #define RAD_ATTR_NAS_IP_ADDR 4 /∗ip∗/

  Radius attribute server IP.
- #define RAD_ATTR_NAS_PORT 5 /∗int∗/

  Radius attribute server port.
- #define RAD_ATTR_SERVICE_TYPE 6 /∗int∗/

  Radius attribute service type.
- #define RAD_ATTR_ACCTID 44

  Radius attribute account id.
- #define RAD_ATTR_PORT_TYPE 61 /∗int∗/

  Radius attribute port type.
- #define RAD_ATTR_EAP 79 /∗oct∗/

  Radius attribute EAP.
- #define RAD_ATTR_MESSAGE 80 /∗oct∗/

  Radius attribute message.

Typedefs

- typedef struct radius_packet radius_packet

  Forward decleration of structure.
- typedef void(∗ radius_cb)(struct radius_packet ∗, void ∗)

  Callback to call when response arrives.

Enumerations

- enum RADIUS_CODE {
  RAD_CODE_AUTHREQUEST = 1, RAD_CODE_AUTHACCEPT = 2, RAD_CODE_AUTHREJECT = 3, RAD_CODE_ACCTREQUEST = 4,
  RAD_CODE_ACCTRESPONSE = 5, RAD_CODE_AUTHCHALLENGE = 11 }

  Radius packet codes.

Functions

- void addradattrint (struct radius_packet ∗packet, char type, unsigned int val)

  Add a integer attribute too the packet.
- void addradattrip (struct radius_packet ∗packet, char type, char ∗ipaddr)

  Add a integer attribute too the packet.
- void addradattrstr (struct radius_packet ∗packet, char type, char ∗str)

  Add a integer attribute too the packet.
- struct radius_packet ∗ new_radpacket (unsigned char code)

  Create a new radius packet.
- void add_radserver (const char ∗ipaddr, const char ∗auth, const char ∗acct, const char ∗secret, int timeout)

  Add new radius server to list of servers.
- int send_radpacket (struct radius_packet ∗packet, const char ∗userpass, radius_cb read_cb, void ∗cb_data)

  Send radius packet.
- unsigned char ∗ radius_attr_first (struct radius_packet ∗packet)

  Return first packet attribute.
- unsigned char ∗ radius_attr_next (struct radius_packet ∗packet, unsigned char ∗attr)

  Return next packet attribute.

12.11.1 Detailed Description

Simple implementation of experimental radius client.

```
* User password crypt function from the freeradius project (addattrpasswd)
* Copyright (C) 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009 The FreeRADIUS Server Projec
```

12.11.2 Macro Definition Documentation

12.11.2.1 #define RAD_ATTR_ACCTID 44

Radius attribute account id.

Definition at line 556 of file dtsapp.h.

12.11.2.2 #define RAD_ATTR_EAP 79 /∗oct∗/

Radius attribute EAP.

Definition at line 562 of file dtsapp.h.

12.11.2.3 #define RAD_ATTR_MESSAGE 80 /*oct*/

Radius attribute message.

Definition at line 565 of file dtsapp.h.

12.11.2.4 #define RAD_ATTR_NAS_IP_ADDR 4 /*ip*/

Radius attribute server IP.

Definition at line 547 of file dtsapp.h.

12.11.2.5 #define RAD_ATTR_NAS_PORT 5 /*int*/

Radius attribute server port.

Definition at line 550 of file dtsapp.h.

12.11.2.6 #define RAD_ATTR_PORT_TYPE 61 /*int*/

Radius attribute port type.

Definition at line 559 of file dtsapp.h.

12.11.2.7 #define RAD_ATTR_SERVICE_TYPE 6 /*int*/

Radius attribute service type.

Definition at line 553 of file dtsapp.h.

12.11.2.8 #define RAD_ATTR_USER_NAME 1 /*string*/

Radius attribute username.

Definition at line 541 of file dtsapp.h.

12.11.2.9 #define RAD_ATTR_USER_PASSWORD 2 /*passwd*/

Radius attribute password.

Definition at line 544 of file dtsapp.h.

12.11.2.10 #define RAD_AUTH_HDR_LEN 20

Authentification header length.

Definition at line 529 of file dtsapp.h.

Referenced by new_radpacket(), and radius_attr_next().

12.11.2.11 #define RAD_AUTH_PACKET_LEN 4096

Auth packet length.

Definition at line 532 of file dtsapp.h.

12.11.2.12 #define RAD_AUTH_TOKEN_LEN 16

Auth token length.

Definition at line 535 of file dtsapp.h.

Referenced by new_radpacket().

12.11.2.13 #define RAD_MAX_PASS_LEN 128

Auth max password length.

Definition at line 538 of file dtsapp.h.

12.11.3 Typedef Documentation

12.11.3.1 typedef void(* radius_cb)(struct radius_packet *, void *)

Callback to call when response arrives.

Parameters

packet	Reference to radius packet.
data	Reference to userdata.

Definition at line 306 of file dtsapp.h.

12.11.3.2 typedef struct radius_packet radius_packet

Forward decleration of structure.

Definition at line 193 of file dtsapp.h.

12.11.4 Enumeration Type Documentation

12.11.4.1 enum RADIUS_CODE

Radius packet codes.

Enumerator

RAD_CODE_AUTHREQUEST Radius auth request.

RAD_CODE_AUTHACCEPT Radius auth accept.

RAD_CODE_AUTHREJECT Radius auth reject.

RAD_CODE_ACCTREQUEST Radius accounting request.

RAD_CODE_ACCTRESPONSE Radius accounting response.

RAD_CODE_AUTHCHALLENGE Radius auth challenge.

Definition at line 568 of file dtsapp.h.

```
00568                 {
00570     RAD_CODE_AUTHREQUEST    =   1,
00572     RAD_CODE_AUTHACCEPT  =   2,
00574     RAD_CODE_AUTHREJECT  =   3,
00576     RAD_CODE_ACCTREQUEST    =   4,
00578     RAD_CODE_ACCTRESPONSE  =   5,
00580     RAD_CODE_AUTHCHALLENGE    =   11
00581 };
```

12.11.5 Function Documentation

12.11.5.1 void add_radserver (const char * *ipaddr*, const char * *auth*, const char * *acct*, const char * *secret*, int *timeout*)

Add new radius server to list of servers.

Parameters

ipaddr	IP address or hostname of server.
auth	Athentification port.
acct	Accounting port.
secret	Shared secret.
timeout	Time to take offline on failure.

Definition at line 289 of file radius.c.

References radius_server::acctport, addtobucket(), ALLOC_CONST, radius_server::authport, bucket_list_cnt(), create_bucketlist(), radius_server::id, radius_server::name, objalloc(), objunref(), radius_server::secret, radius_server::service, and radius_server::timeout.

```
00289
            {
00290     struct radius_server *server;
00291
00292     if ((server = objalloc(sizeof(*server), del_radserver))) {
00293         ALLOC_CONST(server->name, ipaddr);
00294         ALLOC_CONST(server->authport, auth);
00295         ALLOC_CONST(server->acctport, acct);
00296         ALLOC_CONST(server->secret, secret);
00297         if (!servers) {
00298             servers = create_bucketlist(0, hash_server);
00299         }
00300         server->id = bucket_list_cnt(servers);
00301         server->timeout = timeout;
00302         gettimeofday(&server->service, NULL);
00303         addtobucket(servers, server);
00304     }
00305
00306     objunref(server);
00307 }
```

12.11.5.2 void addradattrint (struct **radius_packet** * *packet*, char *type*, unsigned int *val*)

Add a integer attribute too the packet.

Parameters

packet	Radius packet to add too.
type	Attribute been added.
val	Value to add.

Definition at line 149 of file radius.c.

```
00149                                                                    {
00150     unsigned int tval;
00151
00152     tval = htonl(val);
00153     addradattr(packet, type, (unsigned char *)&tval, sizeof(tval));
00154 }
```

12.11.5.3 void addradattrip (struct **radius_packet** * *packet*, char *type*, char * *ipaddr*)

Add a integer attribute too the packet.

Parameters

packet	Radius packet to add too.
type	Attribute been added.
ipaddr	IP to add.

Definition at line 160 of file radius.c.

```
00160                                                                          {
00161     unsigned int tval;
00162
00163     tval = inet_addr(ipaddr);
00164     addradattr(packet, type, (unsigned char *)&tval, sizeof(tval));
00165 }
```

12.11.5.4 void addradattrstr (struct radius_packet * *packet*, char *type*, char * *str*)

Add a integer attribute too the packet.

Parameters

packet	Radius packet to add too.
type	Attribute been added.
str	Value to add.

Definition at line 171 of file radius.c.

```
00171                                                                          {
00172     addradattr(packet, type, (unsigned char *)str, strlen(str));
00173 }
```

12.11.5.5 struct radius_packet* new_radpacket (unsigned char *code*)

Create a new radius packet.

See Also

RADIUS_CODE

Parameters

code	Radius packet type.

Returns

reference to new radius packet of specified type.

Definition at line 221 of file radius.c.

References radius_packet::code, genrand(), radius_packet::len, RAD_AUTH_HDR_LEN, RAD_AUTH_TOKEN_LEN, and radius_packet::token.

```
00221                                                          {
00222     struct radius_packet *packet;
00223
00224     if ((packet = malloc(sizeof(*packet)))) {
00225         memset(packet, 0, sizeof(*packet));
00226         packet->len = RAD_AUTH_HDR_LEN;
00227         packet->code = code;
00228         genrand(&packet->token, RAD_AUTH_TOKEN_LEN);
00229     }
00230     return (packet);
00231 }
```

12.11.5.6 unsigned char* radius_attr_first (struct radius_packet * *packet*)

Return first packet attribute.

Used with radius_attr_next() to iterate through attributes.

Parameters

packet	Radius packet.

Returns

Pointer to next attribute

Definition at line 627 of file radius.c.

References radius_packet::attrs.

```
00627                                                                   {
00628     return (packet->attrs);
00629 }
```

12.11.5.7 unsigned char* radius_attr_next (struct radius_packet * *packet,* unsigned char * *attr*)

Return next packet attribute.

Parameters

packet	Radius packet.
attr	Last attribute.

Returns

Pointer to next attribute.

Definition at line 635 of file radius.c.

References radius_packet::attrs, radius_packet::len, and RAD_AUTH_HDR_LEN.

```
00635                                                                                   {
00636     int offset = (packet->len - RAD_AUTH_HDR_LEN) - (attr - packet->
      attrs);
00637
00638     if (!(offset - attr[1])) {
00639         return NULL;
00640     }
00641
00642     return (attr + attr[1]);
00643 }
```

12.11.5.8 int send_radpacket (struct radius_packet * *packet,* const char * *userpass,* radius_cb *read_cb,* void * *cb_data*)

Send radius packet.

Parameters

packet	Radius packet to send.

userpass	Userpassword if required (added last requires special processing)
read_cb	Callback to call when response arrives.
cb_data	Reference to pass to callback.

Returns

0 on success.

Definition at line 452 of file radius.c.

```
00452
           {
00453     return (_send_radpacket(packet, userpass, NULL, read_cb, cb_data));
00454 }
```

12.12 Micelaneous utilities.

Utilities commonly used.

Modules

- Hashing and digest functions

  MD5/SHA1/SHA2(256/512) Hashing checking and HMAC Functions.

Files

- file util.c

  Utilities commonly used.

Functions

- void seedrand (void)

  Seed openssl random number generator.
- int genrand (void ∗buf, int len)

  Generate random sequence.
- int strlenzero (const char ∗str)

  Check if a string is zero length.
- char ∗ ltrim (char ∗str)

  Trim white space at the begining of a string.
- char ∗ rtrim (const char ∗str)

  Trim white space at the end of a string.
- char ∗ trim (const char ∗str)

  Trim whitesapce from the beggining and end of a string.
- uint64_t tvtontp64 (struct timeval ∗tv)

  Convert a timeval struct to 64bit NTP time.
- uint16_t checksum (const void ∗data, int len)

  Obtain the checksum for a buffer.
- uint16_t checksum_add (const uint16_t checksum, const void ∗data, int len)

  Obtain the checksum for a buffer adding a checksum.
- uint16_t verifysum (const void ∗data, int len, const uint16_t check)

  Verify a checksum.
- void touch (const char ∗filename, uid_t user, gid_t group)

  Create a file and set user and group.
- char ∗ b64enc_buf (const char ∗message, uint32_t len, int nonl)

  Base 64 encode a buffer.
- char ∗ b64enc (const char ∗message, int nonl)

  Base 64 encode a string.

12.12.1 Detailed Description

Utilities commonly used.

12.12.2 Function Documentation

12.12.2.1 char* b64enc (const char * *message*, int *nonl*)

Base 64 encode a string.

Parameters

message	String to encode.
nonl	Encode the data all on one line if non zero.

Returns

Reference to base64 encoded string.

Definition at line 539 of file util.c.

References b64enc_buf().

```
00539                                                   {
00540     return b64enc_buf(message, strlen(message), nonl);
00541 }
```

12.12.2.2 char* b64enc_buf (const char * *message*, uint32_t *len*, int *nonl*)

Base 64 encode a buffer.

Parameters

message	Buffer to encode.
len	Length of the buffer.
nonl	Encode the data all on one line if non zero.

Returns

Reference to base64 encoded string.

Definition at line 506 of file util.c.

References objalloc().

Referenced by b64enc().

```
00506                                                                   {
00507     BIO *bmem, *b64;
00508     BUF_MEM *ptr;
00509     char *buffer;
00510     double encodedSize;
00511
00512     encodedSize = 1.36*len;
00513     buffer = objalloc(encodedSize+1, NULL);
00514
00515     b64 = BIO_new(BIO_f_base64());
00516     bmem = BIO_new(BIO_s_mem());
00517     b64 = BIO_push(b64, bmem);
00518     if (nonl) {
00519         BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL);
00520     }
00521     BIO_write(b64, message, len);
00522     BIO_flush(b64);
00523     BIO_get_mem_ptr(b64, &ptr);
00524
00525     buffer = objalloc(ptr->length+1, NULL);
00526     memcpy(buffer, ptr->data, ptr->length);
00527
00528
00529     BIO_free_all(b64);
00530
00531     return buffer;
00532 }
```

12.12.2.3 uint16_t checksum (const void * *data*, int *len*)

Obtain the checksum for a buffer.

Parameters

data	Buffer to create checksum of.
len	Buffer length.

Returns

Chechsum of data.

Definition at line 452 of file util.c.

Referenced by ipv4checksum(), ipv4icmpchecksum(), ipv4tcpchecksum(), ipv4udpchecksum(), and rfc6296_map_-add().

```
00452                                                   {
00453     return (_checksum(data, len, 0));
00454 }
```

12.12.2.4 uint16_t checksum_add (const uint16_t *checksum*, const void * *data*, int *len*)

Obtain the checksum for a buffer adding a checksum.

Parameters

checksum	Checksum to add to generated checksum.
data	Buffer to create checksum of.
len	Buffer length.

Returns

Chechsum of data.

Definition at line 463 of file util.c.

Referenced by ipv4tcpchecksum(), and ipv4udpchecksum().

```
00463                                                                   {
00464     return (_checksum(data, len, ~checksum));
00465 }
```

12.12.2.5 int genrand (void * *buf*, int *len*)

Generate random sequence.

Parameters

buf	Buffer to write random data.
len	Length to write.

Returns

1 on success 0 otherwise.

Definition at line 82 of file util.c.

Referenced by mcast4_ip(), mcast6_ip(), new_radpacket(), randhwaddr(), and sslstartup().

```
00082                                   {
00083     return (RAND_bytes(buf, len));
00084 }
```

12.12.2.6 char* ltrim (char * *str*)

Trim white space at the begining of a string.

Parameters

str	String to trim.

Returns

Pointer to trimed string.

Definition at line 353 of file util.c.

References strlenzero().

Referenced by trim().

```
00353                                {
00354     char *cur = str;
00355
00356     if (strlenzero(str)) {
00357         return (str);
00358     }
00359
00360     while(isspace(cur[0])) {
00361         cur++;
00362     }
00363
00364     return (cur);
00365 }
```

12.12.2.7 char* rtrim (const char * *str*)

Trim white space at the end of a string.

Parameters

str	String to trim.

Returns

Pointer to trimed string.

Definition at line 372 of file util.c.

References strlenzero().

Referenced by trim().

```
00372                                    {
00373     int len;
00374     char *cur = (char *)str;
00375
00376     if (strlenzero(str)) {
00377         return (cur);
00378     }
00379
00380     len = strlen(str) - 1;
00381     while(len && isspace(cur[len])) {
00382         cur[len] = '\0';
00383         len--;
00384     }
00385
00386     return (cur);
00387 }
```

12.12.2.8 void seedrand (void)

Seed openssl random number generator.

This should be run at application startup

Todo This wont work on WIN32

Definition at line 68 of file util.c.

Referenced by framework_init(), and mcast_socket().

```
00068                               {
00069     int fd = open("/dev/random", O_RDONLY);
00070     int len;
00071     char    buf[64];
00072
00073     len = read(fd, buf, 64);
00074     RAND_seed(buf, len);
00075 }
```

12.12.2.9 int strlenzero (const char * *str*)

Check if a string is zero length.

strlen can not be used on a NULL string this is a quick and dirty util to check it.

Parameters

str	String to check.

Returns

1 if the string is null or zero length

Definition at line 341 of file util.c.

Referenced by create_kernmac(), create_kernvlan(), get_category_next(), get_ifinfo(), get_ifipaddr(), ifhwaddr(), ltrim(), process_config(), rtrim(), and unixsocket_client().

```
00341                                       {
00342     if (str && strlen(str)) {
00343         return (0);
00344     }
00345     return (1);
00346 }
```

12.12.2.10 void touch (const char * *filename*, uid_t *user*, gid_t *group*)

Create a file and set user and group.

Todo WIN32 does not use uid/gid and move to file utils module.

Parameters

filename	File to create.
user	User ID to set ownership.
group	Group ID to set ownership.

Definition at line 484 of file util.c.

References touch().

Referenced by touch(), and xslt_apply().

```
00484                                                                   {
00485     int res;
00486 #else
00487 extern void touch(const char *filename) {
00488 #endif
```

```
00489     int fd;
00490
00491     fd = creat(filename, 0600);
00492     close(fd);
00493 #ifndef __WIN32__
00494     res = chown(filename, user, group);
00495     res++;
00496 #endif
00497     return;
00498 }
```

12.12.2.11 char* trim (const char * *str*)

Trim whitesapce from the beggining and end of a string.

Parameters

str	String to trim.

Returns

Trimed string.

Definition at line 393 of file util.c.

References ltrim(), and rtrim().

Referenced by process_config().

```
00393                                   {
00394     char *cur = (char *)str;
00395
00396     cur = ltrim(cur);
00397     cur = rtrim(cur);
00398     return (cur);
00399 }
```

12.12.2.12 uint64_t tvtontp64 (struct timeval * *tv*)

Convert a timeval struct to 64bit NTP time.

Parameters

tv	Timeval struct to convert.

Returns

64 bit NTP time value.

Definition at line 405 of file util.c.

Referenced by get_ip6_addrprefix().

```
00405                                           {
00406     return ((((uint64_t)tv->tv_sec + 2208988800u) << 32) + ((uint32_t)tv->tv_usec * 4294.967296));
00407 }
```

12.12.2.13 uint16_t verifysum (const void * *data,* int *len,* const uint16_t *check*)

Verify a checksum.

Parameters

data	Data to generate checksum.
len	Length of data.
check	Checksum to check against.

Returns

0 when checksum is verified.

Definition at line 473 of file util.c.

```
00473                                                                   {
00474     return (_checksum(data, len, check));
00475 }
```

12.13 Hashing and digest functions

MD5/SHA1/SHA2(256/512) Hashing checking and HMAC Functions.

Modules

- MD5 Hashing and digest functions

  MD5 Hashing checking and HMAC Functions.
- SHA1 Hashing and digest functions

  SHA1 Hashing checking and HMAC Functions.
- SHA2-256Hashing and digest functions

  SHA2-256 Hashing checking and HMAC Functions.
- SHA2-512 Hashing and digest functions

  SHA2-512 Hashing checking and HMAC Functions.

Files

- file util.c

  Utilities commonly used.

12.13.1 Detailed Description

MD5/SHA1/SHA2(256/512) Hashing checking and HMAC Functions.

```
* Acknowledgments [MD5 HMAC http://www.ietf.org/rfc/rfc2104.txt]
*      Pau-Chen Cheng, Jeff Kraemer, and Michael Oehler, have provided
*      useful comments on early drafts, and ran the first interoperability
*      tests of this specification. Jeff and Pau-Chen kindly provided the
*      sample code and test vectors that appear in the appendix.  Burt
*      Kaliski, Bart Preneel, Matt Robshaw, Adi Shamir, and Paul van
*      Oorschot have provided useful comments and suggestions during the
*      investigation of the HMAC construction.
*
```

12.14 MD5 Hashing and digest functions

MD5 Hashing checking and HMAC Functions.

Functions

- void md5sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the MD5 hash accross 2 data chunks.
- void md5sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the MD5 hash.
- int md5cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two md5 hashes.
- void md5hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) MD5.

12.14.1 Detailed Description

MD5 Hashing checking and HMAC Functions.

12.14.2 Function Documentation

12.14.2.1 int md5cmp (unsigned char * *digest1*, unsigned char * *digest2*)

Compare two md5 hashes.

Parameters

digest1	Digest to compare.
digest2	Digest to compare.

Returns

0 on equality.

Definition at line 223 of file util.c.

```
00223                                                                {
00224     return (_digest_cmp(digest1, digest2, 16));
00225 }
```

12.14.2.2 void md5hmac (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *key*, unsigned long *klen*)

Hash Message Authentication Codes (HMAC) MD5.

Parameters

buff	HMAC returned in this buffer (16 bytes).
data	Data to sign.

len	Length of data.
key	Key to signwith.
klen	Length of key.

Definition at line 290 of file util.c.

References md5sum2().

```
00290
                {
00291     _hmac(buff, data, len, key, klen, md5sum2, 16);
00292 }
```

12.14.2.3 void md5sum (unsigned char * *buff*, const void * *data*, unsigned long *len*)

Calculate the MD5 hash.

Parameters

buff	buffer to place the hash (16 bytes).
data	First data chunk to calculate.
len	Length of data.

Definition at line 202 of file util.c.

References md5sum2().

```
00202                                                                   {
00203     md5sum2(buff, data, len, NULL, 0);
00204 }
```

12.14.2.4 void md5sum2 (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *data2*, unsigned long *len2*)

Calculate the MD5 hash accross 2 data chunks.

Parameters

buff	buffer to place the hash (16 bytes).
data	First data chunk to calculate.
len	Length of data.
data2	Second data chunk to calculate.
len2	Length of data2.

Definition at line 185 of file util.c.

Referenced by md5hmac(), and md5sum().

```
00185
                {
00186     MD5_CTX c;
00187
00188     MD5_Init(&c);
00189     MD5_Update(&c, data, len);
00190     if (data2) {
00191         MD5_Update(&c, data2, len2);
00192     }
00193     MD5_Final(buff, &c);
00194 }
```

12.15 SHA1 Hashing and digest functions

SHA1 Hashing checking and HMAC Functions.

Functions

- void sha1sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA1 hash accross 2 data chunks.
- void sha1sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA1 hash.
- int sha1cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA1 hashes.
- void sha1hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA1.

12.15.1 Detailed Description

SHA1 Hashing checking and HMAC Functions.

12.15.2 Function Documentation

12.15.2.1 int sha1cmp (unsigned char * *digest1*, unsigned char * *digest2*)

Compare two SHA1 hashes.

Parameters

digest1	Digest to compare.
digest2	Digest to compare.

Returns

0 on equality.

Definition at line 233 of file util.c.

```
00233                                                                   {
00234     return (_digest_cmp(digest1, digest2, 20));
00235 }
```

12.15.2.2 void sha1hmac (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *key*, unsigned long *klen*)

Hash Message Authentication Codes (HMAC) SHA1.

Parameters

buff	HMAC returned in this buffer (20 bytes).
data	Data to sign.

len	Length of data.
key	Key to signwith.
klen	Length of key.

Definition at line 302 of file util.c.

References sha1sum2().

```
00302
                {
00303     _hmac(buff, data, len, key, klen, sha1sum2, 20);
00304 }
```

12.15.2.3 void sha1sum (unsigned char * *buff*, const void * *data*, unsigned long *len*)

Calculate the SHA1 hash.

Parameters

buff	buffer to place the hash (20 bytes).
data	First data chunk to calculate.
len	Length of data.

Definition at line 173 of file util.c.

References sha1sum2().

```
00173                                                                   {
00174     sha1sum2(buff, data, len, NULL, 0);
00175 }
```

12.15.2.4 void sha1sum2 (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *data2*, unsigned long *len2*)

Calculate the SHA1 hash accross 2 data chunks.

Parameters

buff	buffer to place the hash (20 bytes).
data	First data chunk to calculate.
len	Length of data.
data2	Second data chunk to calculate.
len2	Length of data2.

Definition at line 156 of file util.c.

Referenced by get_ip6_addrprefix(), sha1hmac(), and sha1sum().

```
00156
                {
00157     SHA_CTX c;
00158
00159     SHA_Init(&c);
00160     SHA_Update(&c, data, len);
00161     if (data2) {
00162         SHA_Update(&c, data2, len2);
00163     }
00164     SHA_Final(buff, &c);
00165 }
```

12.16 SHA2-256Hashing and digest functions

SHA2-256 Hashing checking and HMAC Functions.

Functions

- void sha256sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA2-256 hash accross 2 data chunks.
- void sha256sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA2-256 hash.
- int sha256cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA2-256 hashes.
- void sha256hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA2-256.

12.16.1 Detailed Description

SHA2-256 Hashing checking and HMAC Functions.

12.16.2 Function Documentation

12.16.2.1 int sha256cmp (unsigned char * *digest1*, unsigned char * *digest2*)

Compare two SHA2-256 hashes.

Parameters

digest1	Digest to compare.
digest2	Digest to compare.

Returns

0 on equality.

Definition at line 243 of file util.c.

```
00243                                                                   {
00244     return (_digest_cmp(digest1, digest2, 32));
00245 }
```

12.16.2.2 void sha256hmac (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *key*, unsigned long *klen*)

Hash Message Authentication Codes (HMAC) SHA2-256.

Parameters

buff	HMAC returned in this buffer (32 bytes).

data	Data to sign.
len	Length of data.
key	Key to signwith.
klen	Length of key.

Definition at line 314 of file util.c.

References sha256sum2().

```
00314
                {
00315     _hmac(buff, data, len, key, klen, sha256sum2, 32);
00316 }
```

12.16.2.3 void sha256sum (unsigned char * *buff*, const void * *data*, unsigned long *len*)

Calculate the SHA2-256 hash.

Parameters

buff	buffer to place the hash (32 bytes).
data	First data chunk to calculate.
len	Length of data.

Definition at line 144 of file util.c.

References sha256sum2().

```
00144                                                                   {
00145     sha256sum2(buff, data, len, NULL, 0);
00146 }
```

12.16.2.4 void sha256sum2 (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *data2*, unsigned long *len2*)

Calculate the SHA2-256 hash accross 2 data chunks.

Parameters

buff	buffer to place the hash (32 bytes).
data	First data chunk to calculate.
len	Length of data.
data2	Second data chunk to calculate.
len2	Length of data2.

Definition at line 127 of file util.c.

Referenced by sha256hmac(), and sha256sum().

```
00127
                  {
00128     SHA256_CTX c;
00129
00130     SHA256_Init(&c);
00131     SHA256_Update(&c, data, len);
00132     if (data2) {
00133         SHA256_Update(&c, data2, len2);
00134     }
00135     SHA256_Final(buff, &c);
00136 }
```

12.17 SHA2-512 Hashing and digest functions

SHA2-512 Hashing checking and HMAC Functions.

Functions

- void sha512sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA2-512 hash accross 2 data chunks.
- void sha512sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA2-512 hash.
- int sha512cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA2-512 hashes.
- void sha512hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA2-512.

12.17.1 Detailed Description

SHA2-512 Hashing checking and HMAC Functions.

12.17.2 Function Documentation

12.17.2.1 int sha512cmp (unsigned char * *digest1*, unsigned char * *digest2*)

Compare two SHA2-512 hashes.

Parameters

digest1	Digest to compare.
digest2	Digest to compare.

Returns

0 on equality.

Definition at line 253 of file util.c.

```
00253                                                                   {
00254     return (_digest_cmp(digest1, digest2, 64));
00255 }
```

12.17.2.2 void sha512hmac (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *key*, unsigned long *klen*)

Hash Message Authentication Codes (HMAC) SHA2-512.

Parameters

buff	HMAC returned in this buffer (64 bytes).

data	Data to sign.
len	Length of data.
key	Key to signwith.
klen	Length of key.

Definition at line 326 of file util.c.

References sha512sum2().

```
00326
                {
00327     _hmac(buff, data, len, key, klen, sha512sum2, 64);
00328 }
```

12.17.2.3 void sha512sum (unsigned char * *buff*, const void * *data*, unsigned long *len*)

Calculate the SHA2-512 hash.

Parameters

buff	buffer to place the hash (64 bytes).
data	First data chunk to calculate.
len	Length of data.

Definition at line 114 of file util.c.

References sha512sum2().

```
00114                                                                   {
00115     sha512sum2(buff, data, len, NULL, 0);
00116 }
```

12.17.2.4 void sha512sum2 (unsigned char * *buff*, const void * *data*, unsigned long *len*, const void * *data2*, unsigned long *len2*)

Calculate the SHA2-512 hash accross 2 data chunks.

Parameters

buff	buffer to place the hash (64 bytes).
data	First data chunk to calculate.
len	Length of data.
data2	Second data chunk to calculate.
len2	Length of data2.

Definition at line 97 of file util.c.

Referenced by sha512hmac(), and sha512sum().

```
00097
                  {
00098     SHA512_CTX c;
00099
00100     SHA512_Init(&c);
00101     SHA512_Update(&c, data, len);
00102     if (data2) {
00103         SHA512_Update(&c, data2, len2);
00104     }
00105     SHA512_Final(buff, &c);
00106 }
```

12.18 IPv4 and IPv6 functions

Helper functions for various calculations.

Modules

- IPv4 functions

  Helper functions for various calculations.
- IPv6 functions

  Helper functions for various calculations.

Files

- file iputil.c

  IPv4 And IPv6 Utiliies.

Enumerations

- enum ipversion { IP_PROTO_V4 = 4, IP_PROTO_V6 = 6 }

  IP Protocol numbers.

Functions

- int packetchecksum (uint8_t *pkt)

  Generic IPv4 and IPv6 Checksum.
- int inet_lookup (int family, const char *host, void *addr, socklen_t len)

  Perform DNS lookup on a host/ip retun the IP address.

12.18.1 Detailed Description

Helper functions for various calculations.

12.18.2 Enumeration Type Documentation

12.18.2.1 enum ipversion

IP Protocol numbers.

Enumerator

IP_PROTO_V4

IP_PROTO_V6

Definition at line 77 of file iputil.c.

```
00077                {
00078     IP_PROTO_V4 = 4,
00079     IP_PROTO_V6 = 6
00080 };
```

12.18.3 Function Documentation

12.18.3.1 int inet_lookup (int *family,* const char * *host,* void * *addr,* socklen_t *len*)

Perform DNS lookup on a host/ip retun the IP address.

Parameters

family	Protocol family either PF_INET or PF_INET6.
host	Hostname or IP address to lookup.
addr	A structure in_addr or in6_addr the result is returned in.
len	Length of the structure to place the result.

Returns

0 on failure ie addr is unaltered.

Definition at line 523 of file iputil.c.

Referenced by mcast_socket().

```
00523                                                                                 {
00524     struct addrinfo hint, *result, *ainfo;
00525     int ret = 0;
00526
00527     memset(&hint, 0, sizeof(hint));
00528     hint.ai_family = family;
00529
00530     if (getaddrinfo(host, NULL, &hint, &result) || !result) {
00531         return ret;
00532     }
00533
00534     for(ainfo = result; ainfo; ainfo = ainfo->ai_next) {
00535         switch(ainfo->ai_family) {
00536             case PF_INET:
00537                 if (len >= sizeof(struct in_addr)) {
00538                     struct sockaddr_in *sa4 = (struct sockaddr_in*)ainfo->ai_addr;
00539                     memcpy(addr, &sa4->sin_addr, len);
00540                     ret = 1;
00541                 }
00542                 break;
00543             case PF_INET6:
00544                 if (len >= sizeof(struct in6_addr)) {
00545                     struct sockaddr_in6 *sa6 = (struct sockaddr_in6*)ainfo->ai_addr;
00546                     memcpy(addr, &sa6->sin6_addr, len);
00547                     ret = 1;
00548                 }
00549                 break;
00550         }
00551         if (ret) {
00552             break;
00553         }
00554     }
00555     freeaddrinfo(result);
00556     return ret;
00557 }
```

12.18.3.2 int packetchecksum (uint8_t * *pkt*)

Generic IPv4 and IPv6 Checksum.

Parameters

pkt	Packet buffer to check.

Returns

Checksum.

Definition at line 208 of file iputil.c.

References IP_PROTO_V4, IP_PROTO_V6, and packetchecksumv4().

```
00208                                       {
00209     struct iphdr *ip = (struct iphdr *)pkt;
00210
00211     switch(ip->version) {
00212         case IP_PROTO_V4:
```

```
00213             return (packetchecksumv4(pkt));
00214             break;
00215         case IP_PROTO_V6:
00216             break;
00217     }
00218     return (-1);
00219 }
```

12.19 IPv4 functions

Helper functions for various calculations.

Files

- file iputil.c

  IPv4 And IPv6 Utiliies.

Data Structures

- struct pseudohdr

  IPv4 header structur to cast a packet too.

Functions

- int score_ipv4 (struct sockaddr_in *sa4, char *ipaddr, int iplen)

  Return a score for a IPv4 addrress.
- void ipv4tcpchecksum (uint8_t *pkt)

  Update the TCP checksum of a IPv4 packet.
- void ipv4udpchecksum (uint8_t *pkt)

  Update the UDP checksum of a IPv4 packet.
- void ipv4icmpchecksum (uint8_t *pkt)

  Set the checksup of a IPv4 ICMP packet.
- void ipv4checksum (uint8_t *pkt)

  Set the checksup of a IPv4 Packet.
- int packetchecksumv4 (uint8_t *pkt)

  Update the checksum of a IPv4 packet.
- const char * cidrtosn (int bitlen, char *buf, int size)

  Return the dotted quad notation subnet mask from a CIDR.
- const char * getnetaddr (const char *ipaddr, int cidr, char *buf, int size)

  Return the network address.
- const char * getfirstaddr (const char *ipaddr, int cidr, char *buf, int size)

  Get the first usable address.
- const char * getbcaddr (const char *ipaddr, int cidr, char *buf, int size)

  Return broadcast address.
- const char * getlastaddr (const char *ipaddr, int cidr, char *buf, int size)

  Get the last usable address.
- uint32_t cidrcnt (int bitlen)

  Return the number of IP addresses in a given bitmask.
- int reservedip (const char *ipaddr)

  Check IP against list of reserved IP's.
- int check_ipv4 (const char *ip, int cidr, const char *test)

  Check if a IP address is in a network.

12.19.1 Detailed Description

Helper functions for various calculations.

12.19.2 Function Documentation

12.19.2.1 int check_ipv4 (const char * *ip*, int *cidr*, const char * *test*)

Check if a IP address is in a network.

Note

ipaddr will be truncated to network address based on cidr.

Parameters

ip	Network address to check against.
cidr	Number of bits in the subnet.
test	IP address to check

Returns

0 if test is not in the network ip/cidr.

Definition at line 456 of file iputil.c.

```
00456                                                                   {
00457     uint32_t ip1, ip2;
00458
00459 #ifndef __WIN32
00460     inet_pton(AF_INET, ip, &ip1);
00461     inet_pton(AF_INET, test, &ip2);
00462 #else
00463     ip1 = inet_addr(ip);
00464     ip2 = inet_addr(test);
00465 #endif
00466
00467     ip1 = ntohl(ip1) >> (32-cidr);
00468     ip2 = ntohl(ip2) >> (32-cidr);
00469
00470     if (!(ip1 ^ ip2)) {
00471         return 1;
00472     } else {
00473         return 0;
00474     }
00475 }
```

12.19.2.2 uint32_t cidrcnt (int *bitlen*)

Return the number of IP addresses in a given bitmask.

Parameters

bitlen	Subnet bits (CIDR).

Returns

Number of IP addreses including network and broadcast address.

Definition at line 372 of file iputil.c.

```
00372                                   {
00373     if (bitlen) {
00374         return pow(2, (32-bitlen));
00375     } else {
00376         return 0xFFFFFFFF;
00377     }
00378 }
```

12.19.2.3 const char* cidrtosn (int *bitlen*, char * *buf*, int *size*)

Return the dotted quad notation subnet mask from a CIDR.

Parameters

bitlen	Subnet length bits.
buf	Buffer to copy the subnet address too.
size	Size of buffer.

Returns

pointer to buffer on success or NULL.

Definition at line 228 of file iputil.c.

```
00228                                                                {
00229     uint32_t nm;
00230     uint8_t *nmb = (uint8_t*)&nm;
00231
00232     if (!buf) {
00233         return NULL;
00234     }
00235
00236     if (bitlen) {
00237         nm = ~((1 << (32-bitlen))-1);
00238     } else {
00239         nm = 0;
00240     }
00241
00242     snprintf(buf, size, "%i.%i.%i.%i", nmb[3], nmb[2], nmb[1], nmb[0]);
00243     return buf;
00244 }
```

12.19.2.4 const char* getbcaddr (const char * *ipaddr*, int *cidr*, char * *buf*, int *size*)

Return broadcast address.

Note

ipaddr will be truncated to network address based on cidr.

Parameters

ipaddr	Network address.
cidr	CIDR subnet bit length.
buf	Buffer to copy address too.
size	Length of buffer.

Returns

Pointer to buffer or NULL on error.

Definition at line 319 of file iputil.c.

```
00319                                                                                {
00320     uint32_t ip, mask;
00321     uint8_t *ipb = (uint8_t*)&ip;
00322
00323 #ifndef __WIN32
00324     inet_pton(AF_INET, ipaddr, &ip);
00325 #else
00326     ip = inet_addr(ipaddr);
00327 #endif
00328     if (cidr) {
00329         mask = (1 << (32-cidr))-1;
00330         ip = ntohl(ip);
00331         ip = (ip & ~mask) | mask;
00332     } else {
00333         ip = 0;
00334     }
00335     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00336     return buf;
00337 }
```

12.19.2.5 const char* getfirstaddr (const char * *ipaddr*, int *cidr*, char * *buf*, int *size*)

Get the first usable address.

Note

ipaddr will be truncated to network address based on cidr.

Parameters

ipaddr	Network address.
cidr	Bits in the subnet mask.
buf	Buffer that the result is placed in.
size	Length of buffer.

Returns

Pointer to buf with the result copied to buf.

Definition at line 286 of file iputil.c.

```
00286                                                                                {
00287     uint32_t ip;
00288     uint8_t *ipb = (uint8_t*)&ip;
00289
00290     if (!buf) {
00291         return NULL;
00292     }
00293
00294 #ifndef __WIN32
00295     inet_pton(AF_INET, ipaddr, &ip);
00296 #else
00297     ip = inet_addr(ipaddr);
00298 #endif
00299     if (cidr) {
00300         ip = ntohl(ip);
00301         ip = ip & ~((1 << (32-cidr))-1);
00302         ip++;
00303     } else {
00304         ip = 1;
00305     }
00306
00307     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00308     return buf;
00309 }
```

12.19.2.6 const char* getlastaddr (const char * *ipaddr*, int *cidr*, char * *buf*, int *size*)

Get the last usable address.

Note

ipaddr will be truncated to network address based on cidr.

Parameters

ipaddr	Network address.
cidr	Bits in the subnet mask.
buf	Buffer that the result is placed in.
size	Length of buffer.

Returns

Pointer to buf with the result copied to buf.

Definition at line 347 of file iputil.c.

```
00347                                                                                {
00348     uint32_t ip, mask;
00349     uint8_t *ipb = (uint8_t*)&ip;
00350
00351 #ifndef __WIN32
00352     inet_pton(AF_INET, ipaddr, &ip);
00353 #else
00354     ip = inet_addr(ipaddr);
00355 #endif
00356     if (cidr) {
00357         mask = (1 << (32-cidr))-1;
00358         ip = ntohl(ip);
00359         ip = (ip & ~mask) | mask;
00360         ip--;
00361     } else {
00362         ip = 0;
00363     }
00364     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00365     return buf;
00366 }
```

12.19.2.7 const char* getnetaddr (const char * *ipaddr*, int *cidr*, char * *buf*, int *size*)

Return the network address.

Note

ipaddr will be truncated to network address based on cidr.

Parameters

ipaddr	Ipaddr to calculate for
cidr	Length of the subnet bitmask.
buf	Buffer that the result is placed in.
size	Length of buffer.

Returns

Pointer to buf with the result copied to buf.

Definition at line 254 of file iputil.c.

```
00254                                                                                {
00255     uint32_t ip;
00256     uint8_t *ipb = (uint8_t*)&ip;
00257
00258     if (!buf) {
00259         return NULL;
00260     }
00261
00262 #ifndef __WIN32
00263     inet_pton(AF_INET, ipaddr, &ip);
00264 #else
00265     ip = inet_addr(ipaddr);
00266 #endif
00267     if (cidr) {
00268         ip = ntohl(ip);
00269         ip = ip & ~((1 << (32-cidr))-1);
00270     } else {
00271         ip = 0;
00272     }
00273
00274     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00275     return buf;
00276 }
```

12.19.2.8 void ipv4checksum (uint8_t * *pkt*)

Set the checksup of a IPv4 Packet.

Parameters

pkt	Packet to update.

Definition at line 154 of file iputil.c.

References checksum().

Referenced by packetchecksumv4().

```
00154                                       {
00155     struct iphdr *ip = (struct iphdr *)pkt;
00156
00157     ip->check = 0;
00158     ip->check = checksum(ip, (4 * ip->ihl));
00159 }
```

12.19.2.9 void ipv4icmpchecksum (uint8_t * *pkt*)

Set the checksup of a IPv4 ICMP packet.

Parameters

pkt	ICMP Packet to update.

Definition at line 143 of file iputil.c.

References checksum().

Referenced by packetchecksumv4().

```
00143                                           {
00144     struct iphdr *ip = (struct iphdr *)pkt;
00145     struct icmphdr *icmp = (struct icmphdr *)(pkt + (4 * ip->ihl));
00146
00147     icmp->checksum = 0;
00148     icmp->checksum = checksum(icmp, ntohs(ip->tot_len) - (ip->ihl *4));
00149 }
```

12.19.2.10 void ipv4tcpchecksum (uint8_t * *pkt*)

Update the TCP checksum of a IPv4 packet.

Parameters

pkt	Packet to update TCP checksum.

Definition at line 101 of file iputil.c.

References checksum(), checksum_add(), pseudohdr::daddr, pseudohdr::len, pseudohdr::proto, pseudohdr::saddr, and pseudohdr::zero.

Referenced by packetchecksumv4().

```
00101                                           {
00102     struct iphdr *ip = (struct iphdr *)pkt;
00103     struct tcphdr *tcp = (struct tcphdr *)(pkt + (4 * ip->ihl));
00104     uint16_t plen, csum;
00105     struct pseudohdr phdr;
00106
00107     /* get tcp packet len*/
00108     plen = ntohs(ip->tot_len) - (4 * ip->ihl);
00109     tcp->check = 0;
00110     phdr.saddr = ip->saddr;
00111     phdr.daddr = ip->daddr;
00112     phdr.zero = 0;
00113     phdr.proto = ip->protocol;
00114     phdr.len = htons(plen);
00115     csum = checksum(&phdr, sizeof(phdr));
00116     tcp->check = checksum_add(csum, tcp, plen);
00117 }
```

12.19.2.11 void ipv4udpchecksum (uint8_t * *pkt*)

Update the UDP checksum of a IPv4 packet.

Parameters

pkt	Packet to update UDP checksum.

Definition at line 122 of file iputil.c.

References checksum(), checksum_add(), pseudohdr::daddr, pseudohdr::len, pseudohdr::proto, pseudohdr::saddr, and pseudohdr::zero.

Referenced by packetchecksumv4().

```
00122                                           {
00123     struct iphdr *ip = (struct iphdr *)pkt;
00124     struct udphdr *udp = (struct udphdr *)(pkt + (4 * ip->ihl));
00125     uint16_t csum, plen;
00126     struct pseudohdr phdr;
00127
00128     /* get tcp packet len*/
00129     plen = ntohs(ip->tot_len) - (4 * ip->ihl);
00130     udp->check = 0;
00131     phdr.saddr = ip->saddr;
00132     phdr.daddr = ip->daddr;
00133     phdr.zero = 0;
00134     phdr.proto = ip->protocol;
00135     phdr.len = htons(plen);
00136     csum = checksum(&phdr, sizeof(phdr));
00137     udp->check = checksum_add(csum, udp, plen);
00138 }
```

12.19.2.12 int packetchecksumv4 (uint8_t * *pkt*)

Update the checksum of a IPv4 packet.

Parameters

pkt	Packet buffer to update check.

Returns

0 on success.

Definition at line 165 of file iputil.c.

References ipv4checksum(), ipv4icmpchecksum(), ipv4tcpchecksum(), and ipv4udpchecksum().

Referenced by packetchecksum().

```
00165                                           {
00166     struct iphdr *ip = (struct iphdr *)pkt;
00167
00168     ipv4checksum(pkt);
00169
00170     switch(ip->protocol) {
00171         case IPPROTO_ICMP:
00172             ipv4icmpchecksum(pkt);
00173             break;
00174         case IPPROTO_TCP:
00175             ipv4tcpchecksum(pkt);
00176             break;
00177         case IPPROTO_UDP:
00178             ipv4udpchecksum(pkt);
00179             break;
00180         default:
00181             return (-1);
00182     }
00183     return (0);
00184 }
```

12.19.2.13 int reservedip (const char * *ipaddr*)

Check IP against list of reserved IP's.

Parameters

ipaddr	IP addr to check.

Returns

1 if its a private/resrved/not routed IP

Definition at line 384 of file iputil.c.

Referenced by score_ipv4().

```
00384                                           {
00385     uint32_t ip;
00386
00387 #ifndef __WIN32
00388     inet_pton(PF_INET, ipaddr, &ip);
00389 #else
00390     ip = inet_addr(ipaddr);
00391 #endif
00392
00393     ip = ntohl(ip);
00394
00395     if (!((0xe0000000 ^ ip) >> 28)) { /* 224/4*/
00396         return 1;
00397     } else if (!((0x00000000 ^ ip) >> 24)) { /* 0/8 */
00398         return 1;
00399     } else if (!((0x0a000000 ^ ip) >> 24)) { /* 10/8 */
00400         return 1;
00401     } else if (!((0x7f000000 ^ ip) >> 24)) { /* 127/8 */
00402         return 1;
00403     } else if (!((0x64400000 ^ ip) >> 22)) { /* 100.64/10 */
00404         return 1;
00405     } else if (!((0xac100000 ^ ip) >> 20)) { /* 172.16/12 */
00406         return 1;
00407     } else if (!((0xc6120000 ^ ip) >> 17)) { /* 198.18/15 */
00408         return 1;
00409     } else if (!((0xc0a80000 ^ ip) >> 16)) { /* 192.168/16 */
00410         return 1;
00411     } else if (!((0xa9fe0000 ^ ip) >> 16)) { /* 169.254/16 */
00412         return 1;
00413     } else if (!((0xc0000200 ^ ip) >> 8)) { /* 192.0.2/24 */
00414         return 1;
00415     } else if (!((0xc6336400 ^ ip) >> 8)) { /* 198.51.100/24 */
00416         return 1;
00417     } else if (!((0xcb007100 ^ ip) >> 8)) { /* 203.0.113/24 */
00418         return 1;
00419     }
00420     return 0;
00421 }
```

12.19.2.14 int score_ipv4 (struct sockaddr_in * *sa4*, char * *ipaddr*, int *iplen*)

Return a score for a IPv4 addrress.

Note

This does not follow the RFC as gettaddrinfo would.

Parameters

sa4	Socket addr to check.
ipaddr	Buffer to place IP address.
iplen	Length of IP buffer.

Returns

Score based on the IP address Highest is "routable" lowest is Zeroconf.

Definition at line 718 of file interface.c.

References inet_ntop(), IPV4_SCORE_RESERVED, IPV4_SCORE_ROUTABLE, IPV4_SCORE_ZEROCONF, and reservedip().

Referenced by get_ifinfo(), and get_ifipaddr().

```
00718                                                                   {
00719     uint32_t addr;
00720     int nscore;
00721
00722     addr = sa4->sin_addr.s_addr;
00723
00724     /* Get ipaddr string*/
00725     inet_ntop(AF_INET, &sa4->sin_addr, ipaddr, iplen);
00726
00727     /* Score the IP*/
00728     if (!((0xa9fe0000 ^  ntohl(addr)) >> 16)) {
00729         nscore = IPV4_SCORE_ZEROCONF;
00730     } else if (reservedip(ipaddr)) {
00731         nscore = IPV4_SCORE_RESERVED;
00732     } else {
00733         nscore = IPV4_SCORE_ROUTABLE;
00734     }
00735
00736     return nscore;
00737 }
```

12.20 IPv6 functions

Helper functions for various calculations.

Files

- file iputil.c

  IPv4 And IPv6 Utiliies.

Functions

- void eui48to64 (unsigned char *mac48, unsigned char *eui64)

  Generate IPv6 address from mac address.
- int get_ip6_addrprefix (const char *iface, unsigned char *prefix)

  Generate Unique Local IPv6 Unicast Addresses RFC 4193.
- int score_ipv6 (struct sockaddr_in6 *sa6, char *ipaddr, int iplen)

  Return a score for a IPv6 addrress.
- int checkipv6mask (const char *ipaddr, const char *network, uint8_t bits)

  Check if ipaddr is in a network.
- int packetchecksumv6 (uint8_t *pkt)

  Prototype to check checksup on packet.
- char * ipv6to4prefix (const char *ipaddr)

  Return IPv6 to IPv4 Prefix fot the address.

12.20.1 Detailed Description

Helper functions for various calculations.

12.20.2 Function Documentation

12.20.2.1 int checkipv6mask (const char * *ipaddr*, const char * *network*, uint8_t *bits*)

Check if ipaddr is in a network.

Parameters

ipaddr	To check.
network	Network to check against.
bits	Network length.

Returns

0 if the ipaddr is in the network.

Definition at line 47 of file iputil.c.

```
00047                                                                   {
00048     uint8_t cnt, bytelen, bitlen;
00049     uint32_t mask, res = 0;
00050     uint32_t *nw = (uint32_t *)network;
00051     uint32_t *ip = (uint32_t *)ipaddr;
00052
00053     /*calculate significant bytes and bits outside boundry*/
00054     if ((bitlen = bits % 32)) {
00055         bytelen = (bits - bitlen) / 32;
00056         bytelen++;
```

```
00057     } else {
00058         bytelen = bits / 32;
00059     }
00060
00061     /*end loop on first mismatch do not check last block*/
00062     for(cnt = 0; (!res && (cnt < (bytelen - 1))); cnt++) {
00063         res += nw[cnt] ^ ip[cnt];
00064     }
00065
00066     /*process last block if no error sofar*/
00067     if (!res) {
00068         mask = (bitlen) ? htonl(~((1 << (32 - bitlen)) - 1)) : -1;
00069         res += (nw[cnt] & mask) ^ (ip[cnt] & mask);
00070     }
00071
00072     return (res);
00073 }
```

12.20.2.2 void eui48to64 (unsigned char * *mac48*, unsigned char * *eui64*)

Generate IPv6 address from mac address.

this method is sourced from the following IEEE publication Guidelines for 64-bit Global Identifier (EUI-64TM) Registration Authority mac48 is char[ETH_ALEN] eui64 is char[8]

Parameters

mac48	Buffer containing MAC address 6 bytes.
eui64	Buffer that will be written with address 8bytes.

Definition at line 668 of file interface.c.

Referenced by get_ip6_addrprefix().

```
00668                                                                   {
00669     eui64[0] = (mac48[0] & 0xFE) ^ 0x02; /*clear multicast bit and flip local asignment*/
00670     eui64[1] = mac48[1];
00671     eui64[2] = mac48[2];
00672     eui64[3] = 0xFF;
00673     eui64[4] = 0xFE;
00674     eui64[5] = mac48[3];
00675     eui64[6] = mac48[4];
00676     eui64[7] = mac48[5];
00677 }
```

12.20.2.3 int get_ip6_addrprefix (const char * *iface*, unsigned char * *prefix*)

Generate Unique Local IPv6 Unicast Addresses RFC 4193.

Todo WIN32 support

Parameters

iface	External system interface name.
prefix	A buffer char[6] that will contain the prefix.

Returns

-1 on error.

Definition at line 687 of file interface.c.

References eui48to64(), ifhwaddr(), sha1sum2(), and tvtontp64().

```
00687                                                                   {
00688     uint64_t ntpts;
00689     unsigned char eui64[8];
```

```
00690     unsigned char sha1[20];
00691     unsigned char mac48[ETH_ALEN];
00692     struct timeval tv;
00693 
00694     if (ifhwaddr(iface, mac48)) {
00695         return (-1);
00696     }
00697 
00698     gettimeofday(&tv, NULL);
00699     ntpts = tvtontp64(&tv);
00700 
00701     eui48to64(mac48, eui64);
00702     sha1sum2(sha1, (void *)&ntpts, sizeof(ntpts), (void *)eui64, sizeof(eui64));
00703 
00704     prefix[0] = 0xFD; /*0xFC | 0x01 FC00/7 with local bit set [8th bit]*/
00705     memcpy(prefix + 1, sha1+15, 5); /*LSD 40 bits of the SHA hash*/
00706 
00707     return (0);
00708 }
```

12.20.2.4 char* ipv6to4prefix (const char * *ipaddr*)

Return IPv6 to IPv4 Prefix fot the address.

Parameters

ipaddr	IPv4 Address to obtain mapping for

Returns

6to4 Address prefix.

Definition at line 427 of file iputil.c.

```
00427                                                {
00428     uint32_t ip;
00429     uint8_t *ipa;
00430     char *pre6;
00431 
00432 #ifndef __WIN32
00433     if (!inet_pton(AF_INET, ipaddr, &ip)) {
00434         return NULL;
00435     }
00436 #else
00437     if (!(ip = inet_addr(ipaddr))) {
00438         return NULL;
00439     }
00440 #endif
00441 
00442     pre6 = malloc(10);
00443     ipa=(uint8_t*)&ip;
00444     snprintf(pre6, 10, "%02x%02x:%02x%02x", ipa[0], ipa[1], ipa[2], ipa[3]);
00445     return pre6;
00446 }
```

12.20.2.5 int packetchecksumv6 (uint8_t * *pkt*)

Prototype to check checksup on packet.

Parameters

pkt	Packet buffer to check.

Definition at line 189 of file iputil.c.

```
00189                                        {
00190     struct iphdr *ip = (struct iphdr *)pkt;
00191     switch(ip->protocol) {
00192         case IPPROTO_ICMP:
00193             break;
00194         case IPPROTO_TCP:
00195             break;
```

```
00196         case IPPROTO_UDP:
00197             break;
00198         default:
00199             return (-1);
00200     }
00201     return (0);
00202 }
```

12.20.2.6 int score_ipv6 (struct sockaddr_in6 * *sa6,* char * *ipaddr,* int *iplen*)

Return a score for a IPv6 addrress.

Note

This does not follow the RFC as gettaddrinfo would.

Parameters

sa6	Socket addr to check.
ipaddr	Buffer to place IP address.
iplen	Length of IP buffer.

Returns

Score based on the IP address Highest is "routable" lowest is Internal allocation.

Definition at line 746 of file interface.c.

References inet_ntop(), IPV6_SCORE_RESERVED, IPV6_SCORE_ROUTABLE, and IPV6_SCORE_SIXIN4.

Referenced by get_ifinfo(), and get_ifipaddr().

```
00746                                                                   {
00747     uint32_t *ipptr, match;
00748     int nscore;
00749
00750 #ifndef __WIN32
00751     ipptr = sa6->sin6_addr.s6_addr32;
00752 #else
00753     ipptr = (uint32_t*)sa6->sin6_addr.u.Word;
00754 #endif
00755     match = ntohl(ipptr[0]) >> 16;
00756
00757     /* exclude link local multicast and special addresses */
00758     if (!(0xFE80 ^ match) || !(0xFF ^ (match >> 8)) || !match) {
00759         return 0;
00760     }
00761
00762     /*Score ip private/sixin4/routable*/
00763     if (!(0xFC ^ (match >> 9))) {
00764         nscore = IPV6_SCORE_RESERVED;
00765     } else if (match == 2002) {
00766         nscore = IPV6_SCORE_SIXIN4;
00767     } else {
00768         nscore = IPV6_SCORE_ROUTABLE;
00769     }
00770     inet_ntop(AF_INET6, ipptr, ipaddr, iplen);
00771
00772     return nscore;
00773 }
```

12.21 File utility functions

Convinece wrappers arround stat.

Files

- file fileutil.c

  File utilities to test files (fstat)

Functions

- int is_file (const char *path)

  Determine if a file exists.
- int is_dir (const char *path)

  Determine if a path is a directory.
- int is_exec (const char *path)

  Determine if a file is executable.
- int mk_dir (const char *dir, mode_t mode, uid_t user, gid_t group)

  Create a directory.

12.21.1 Detailed Description

Convinece wrappers arround stat.

12.21.2 Function Documentation

12.21.2.1 int is_dir (const char * *path*)

Determine if a path is a directory.

Parameters

path	Path of directory to check.

Returns

1 if the path exists and is a directory 0 othewise.

Definition at line 55 of file fileutil.c.

```
00055                                       {
00056     struct stat sr;
00057     if (!stat(path, &sr) && S_ISDIR(sr.st_mode)) {
00058         return 1;
00059     } else {
00060         return 0;
00061     }
00062 }
```

12.21.2.2 int is_exec (const char * *path*)

Determine if a file is executable.

Parameters

path	Path of file to check.

Returns

1 if the path exists and is executable 0 othewise.

Definition at line 67 of file fileutil.c.

```
00067                                       {
00068     struct stat sr;
00069     if (!stat(path, &sr) && (S_IXUSR & sr.st_mode)) {
00070         return 1;
00071     } else {
00072         return 0;
00073     }
00074 }
```

12.21.2.3 int is_file (const char * *path*)

Determine if a file exists.

Parameters

path	Filename.

Returns

1 if the file exists 0 othewise.

Definition at line 43 of file fileutil.c.

```
00043                                       {
00044     struct stat sr;
00045     if (!stat(path, &sr)) {
00046         return 1;
00047     } else {
00048         return 0;
00049     }
00050 }
```

12.21.2.4 int mk_dir (const char * *dir*, mode_t *mode*, uid_t *user*, gid_t *group*)

Create a directory.

On *NIX systems a mode, uid and gid can be used to set initial permisions.

Parameters

dir	Directory to create.
mode	Initial mode to set.
user	Initial UID.
group	Initial GID.

Returns

non 0 on success on failure the directory may be created but no ownership not set.

Definition at line 87 of file fileutil.c.

```
00087                                                                   {
00088 #endif
00089     struct stat sr;
00090
00091 #ifdef __WIN32
00092     if ((stat(dir, &sr) && (errno == ENOENT)) && !mkdir(dir)) {
00093 #else
00094     if ((stat(dir, &sr) && (errno == ENOENT)) && !mkdir(dir, mode) && !chown(dir, user, group)) {
00095 #endif
00096         return 0;
00097     }
00098     return -1;
00099 }
```

12.22 Openldap/SASL Interface

Functions to interface with a LDAP server.

Files

- file openldap.c
  Openldap/SASL Implementation.

Data Structures

- struct ldap_rdn
  LDAP Relative distingushed name linked list.
- struct ldap_attrval
  LDAP attribute value.
- struct ldap_attr
  LDAP attirbute.
- struct ldap_entry
  LDAP entry.
- struct ldap_results
  LDAP results.
- struct sasl_defaults
  SASL Paramaters used in authentification.
- struct ldap_simple
  LDAP Simple bind.
- struct ldap_conn
  LDAP connection.
- struct ldap_modify
  LDAP Modify structure.
- struct ldap_add
  LDAP Add structure.
- struct ldap_modval
  Linked list of mod values.
- struct ldap_modreq
  LDAP mod request.

Typedefs

- typedef struct ldap_conn ldap_conn
  Forward decleration of structure.
- typedef struct ldap_modify ldap_modify
  Forward decleration of structure.
- typedef struct ldap_add ldap_add
  Forward decleration of structure.

Enumerations

- enum ldap_starttls { LDAP_STARTTLS_NONE, LDAP_STARTTLS_ATTEMPT, LDAP_STARTTLS_ENFORCE }
  SSL connection requirements.
- enum ldap_attrtype { LDAP_ATTRTYPE_CHAR, LDAP_ATTRTYPE_B64, LDAP_ATTRTYPE_OCTET }
  LDAP attribute types.

Functions

- struct ldap_conn * ldap_connect (const char *uri, enum ldap_starttls starttls, int timelimit, int limit, int debug, int *err)

  Connect to a LDAP server.
- int ldap_simplebind (struct ldap_conn *ld, const char *dn, const char *passwd)

  Bind to the connection with simple bind requireing a distingushed name and password.
- int ldap_simplerebind (struct ldap_conn *ldap, const char *initialdn, const char *initialpw, const char *base, const char *filter, const char *uidrdn, const char *uid, const char *passwd)

  Bind to LDAP connection using rebind.
- int ldap_saslbind (struct ldap_conn *ld, const char *mech, const char *realm, const char *authcid, const char *passwd, const char *authzid)

  Bind to the server with SASL.
- const char * ldap_errmsg (int res)

  Return LDAP error for a ldap error.
- struct ldap_results * ldap_search_sub (struct ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res,...)

  Search LDAP connection subtree.
- struct ldap_results * ldap_search_one (struct ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res,...)

  Search LDAP connection one level.
- struct ldap_results * ldap_search_base (struct ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res,...)

  Search LDAP connection base.
- void ldap_unref_attr (struct ldap_entry *entry, struct ldap_attr *attr)

  Remove a attribute from a entry.
- void ldap_unref_entry (struct ldap_results *results, struct ldap_entry *entry)

  Remove a entry from a result.
- struct ldap_entry * ldap_getentry (struct ldap_results *results, const char *dn)

  Find and return the entry from the results for a specific dn.
- struct ldap_attr * ldap_getattr (struct ldap_entry *entry, const char *attr)

  Find and return attribute in a entry.
- struct ldap_modify * ldap_modifyinit (const char *dn)

  Create a modification reference for a DN.
- int ldap_mod_del (struct ldap_modify *lmod, const char *attr,...)

  Delete values from a attribute.
- int ldap_mod_add (struct ldap_modify *lmod, const char *attr,...)

  Add values to a attribute.
- int ldap_mod_rep (struct ldap_modify *lmod, const char *attr,...)

  Replace a attribute.
- int ldap_domodify (struct ldap_conn *ld, struct ldap_modify *lmod)

  Apply the modification to the server.
- int ldap_mod_delattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Delete a value from a attribute in a DN.
- int ldap_mod_remattr (struct ldap_conn *ldap, const char *dn, const char *attr)

  Delete a attribute from a DN.
- int ldap_mod_addattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Add a value for a attribute in a DN.
- int ldap_mod_repattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Replace the value of a attribute in a DN.
- struct ldap_add * ldap_addinit (const char *dn)

  Create a reference to add a new DN.

- int ldap_add_attr (struct ldap_add *ladd, const char *attr,...)
  Add a attribute to new DN.
- int ldap_doadd (struct ldap_conn *ld, struct ldap_add *ladd)
  Write new DN to server.

12.22.1 Detailed Description

Functions to interface with a LDAP server.

12.22.2 Typedef Documentation

12.22.2.1 typedef struct ldap_add ldap_add

Forward decleration of structure.

Definition at line 789 of file dtsapp.h.

12.22.2.2 typedef struct ldap_conn ldap_conn

Forward decleration of structure.

Definition at line 785 of file dtsapp.h.

12.22.2.3 typedef struct ldap_modify ldap_modify

Forward decleration of structure.

Definition at line 787 of file dtsapp.h.

12.22.3 Enumeration Type Documentation

12.22.3.1 enum ldap_attrtype

LDAP attribute types.

Enumerator

LDAP_ATTRTYPE_CHAR Plain text.

LDAP_ATTRTYPE_B64 Base64 encoded.

LDAP_ATTRTYPE_OCTET Binary data.

Definition at line 707 of file dtsapp.h.

```
00707                   {
00709     LDAP_ATTRTYPE_CHAR,
00711     LDAP_ATTRTYPE_B64,
00713     LDAP_ATTRTYPE_OCTET
00714 };
```

12.22.3.2 enum ldap_starttls

SSL connection requirements.

Enumerator

LDAP_STARTTLS_NONE SSL not attempted at all.

LDAP_STARTTLS_ATTEMPT SSL attempted but not required.

LDAP_STARTTLS_ENFORCE SSL is required.

Definition at line 697 of file dtsapp.h.

```
00697                   {
00699     LDAP_STARTTLS_NONE,
00701     LDAP_STARTTLS_ATTEMPT,
00703     LDAP_STARTTLS_ENFORCE
00704 };
```

12.22.4 Function Documentation

12.22.4.1 int ldap_add_attr (struct ldap_add * *ladd*, const char * *attr*, ...)

Add a attribute to new DN.

Parameters

ladd	Reference to new DN structure.
attr	Attribute to add.
...	NULL terminated list of values.

Returns

0 on success.

Definition at line 1500 of file openldap.c.

References objunref().

```
01500                                                                   {
01501     va_list a_list;
01502     char *val;
01503     struct ldap_modreq *modr;
01504
01505     if (!(modr = getaddreq(ladd, attr))) {
01506         return 1;
01507     }
01508
01509     va_start(a_list, attr);
01510     while((val = va_arg(a_list, void *))) {
01511         if (add_modifyval(modr, val)) {
01512             objunref(modr);
01513             return(1);
01514         }
01515     }
01516
01517     objunref(modr);
01518     va_end(a_list);
01519     return 0;
01520 }
```

12.22.4.2 struct ldap_add* ldap_addinit (const char * *dn*)

Create a reference to add a new DN.

Parameters

dn	DN to be created.

Returns

Reference to a structure to configure for adding a new dn.

Definition at line 1462 of file openldap.c.

References ALLOC_CONST, ldap_add::bl, create_bucketlist(), ldap_add::dn, objalloc(), and objunref().

```
01462                                                    {
01463     struct ldap_add *mod;
01464
01465     if (!(mod = objalloc(sizeof(*mod), free_add))) {
01466         return NULL;
01467     }
01468
01469     ALLOC_CONST(mod->dn, dn);
01470     if (!mod->dn) {
01471         objunref(mod);
01472         return NULL;
01473     }
01474
01475     if (!(mod->bl = create_bucketlist(4, modify_hash))) {
01476         objunref(mod);
01477         return NULL;
01478     }
01479
01480     return mod;
01481 }
```

12.22.4.3 struct ldap_conn* ldap_connect (const char * *uri*, enum ldap_starttls *starttls*, int *timelimit*, int *limit*, int *debug*, int * *err*)

Connect to a LDAP server.

Parameters

uri	Server to connect too.
starttls	Starttls flags to disallow,allow or enforce SSL.
timelimit	Query timelimit.
limit	Results limit.
debug	Set LDAP_OPT_DEBUG_LEVEL and LBER_OPT_DEBUG_LEVEL to this level.
err	Pointer to a int that will contain the ldap error on failure.

Returns

Reference to LDAP connection if its NULL the error is returned in err.

Definition at line 335 of file openldap.c.

References ldap_conn::ldap, LDAP_STARTTLS_ENFORCE, LDAP_STARTTLS_NONE, ldap_conn::limit, objalloc(), objunref(), ldap_conn::sasl, ldap_conn::sctrlsp, ldap_conn::timelim, and ldap_conn::uri.

```
00335
                                {
00336     struct ldap_conn *ld;
00337     int version = 3;
00338     int res, sslres;
00339     struct timeval timeout;
00340
00341     if (!(ld = objalloc(sizeof(*ld), free_ldapconn))) {
00342         return NULL;
00343     }
00344
00345     ld->uri = strdup(uri);
00346     ld->sctrlsp = NULL;
00347     ld->timelim = timelimit;
00348     ld->limit = limit;
00349     ld->sasl = NULL;
00350
00351     if ((res = ldap_initialize(&ld->ldap, ld->uri) != LDAP_SUCCESS)) {
00352         objunref(ld);
00353         ld = NULL;
```

```
00354     } else {
00355         if (debug) {
00356             ldap_set_option(NULL, LDAP_OPT_DEBUG_LEVEL, &debug);
00357             ber_set_option(NULL, LBER_OPT_DEBUG_LEVEL, &debug);
00358         }
00359         if (timelimit) {
00360             timeout.tv_sec = timelimit;
00361             timeout.tv_usec = 0;
00362             ldap_set_option(ld->ldap, LDAP_OPT_NETWORK_TIMEOUT, (void *)&timeout);
00363         }
00364         ldap_set_option(ld->ldap, LDAP_OPT_PROTOCOL_VERSION, &version);
00365         ldap_set_option(ld->ldap, LDAP_OPT_REFERRALS, (void *)LDAP_OPT_ON);
00366         ldap_set_rebind_proc(ld->ldap, ldap_rebind_proc, ld);
00367 
00368         if ((starttls != LDAP_STARTTLS_NONE) & !ldap_tls_inplace(ld->
      ldap) && (sslres = ldap_start_tls_s(ld->ldap, ld->sctrlsp, NULL))) {
00369             if (starttls == LDAP_STARTTLS_ENFORCE) {
00370                 objunref(ld);
00371                 ld = NULL;
00372                 res = sslres;
00373             }
00374         }
00375     }
00376     *err = res;
00377     return ld;
00378 }
```

12.22.4.4 int ldap_doadd (struct ldap_conn * *ld*, struct ldap_add * *ladd*)

Write new DN to server.

Parameters

ld	Reference to connection to the LDAP server.
ladd	Reference to new DN to commit to server.

Returns

non zero LDAP error on failure.

Definition at line 1526 of file openldap.c.

References ldap_add::bl, bucket_list_cnt(), ldap_add::dn, init_bucket_loop(), ldap_conn::ldap, next_bucket_loop(), objlock(), objunlock(), objunref(), and ldap_conn::sctrlsp.

```
01526                                                                   {
01527     struct bucket_loop *bloop;
01528     struct ldap_modreq *modr;
01529     LDAPMod **modarr, **tmp, *item;
01530     int tot=0, res;
01531 
01532     tot = bucket_list_cnt(ladd->bl);
01533     tmp = modarr = calloc(sizeof(void *), (tot+1));
01534 
01535     bloop = init_bucket_loop(ladd->bl);
01536     while(bloop && ((modr = next_bucket_loop(bloop)))) {
01537         if (!(item = ldap_reqtoarr(modr, -1))) {
01538             ldap_mods_free(modarr, 1);
01539             return LDAP_NO_MEMORY;
01540         }
01541         *tmp = item;
01542         tmp++;
01543         objunref(modr);
01544     }
01545     objunref(bloop);
01546     *tmp = NULL;
01547 
01548     objlock(ld);
01549     res = ldap_modify_ext_s(ld->ldap, ladd->dn, modarr, ld->sctrlsp, NULL);
01550     objunlock(ld);
01551     ldap_mods_free(modarr, 1);
01552 
01553     return res;
01554 }
```

12.22.4.5 int ldap_domodify (struct ldap_conn * *ld,* struct ldap_modify * *lmod*)

Apply the modification to the server.

Parameters

ld	Reference to LDAP connection.
lmod	Reference to modification structure.

Returns

Non zero ldap error on error.

Definition at line 1339 of file openldap.c.

References ldap_modify::bl, bucket_list_cnt(), ldap_modreq::cnt, ldap_modify::dn, init_bucket_loop(), ldap_conn::ldap, next_bucket_loop(), objlock(), objref(), objunlock(), objunref(), and ldap_conn::sctrlsp.

Referenced by ldap_mod_addattr(), ldap_mod_delattr(), and ldap_mod_repattr().

```
01339                                                                                {
01340     struct bucket_loop *bloop;
01341     struct ldap_modreq *modr;
01342     LDAPMod **modarr, **tmp, *item;
01343     int cnt, tot=0, res;
01344
01345     if (!objref(ld)) {
01346         return LDAP_UNAVAILABLE;
01347     }
01348
01349     for(cnt = 0; cnt < 3; cnt++) {
01350         tot += bucket_list_cnt(lmod->bl[cnt]);
01351     }
01352     tmp = modarr = calloc(sizeof(void *), (tot+1));
01353
01354     for(cnt = 0; cnt < 3; cnt++) {
01355         bloop = init_bucket_loop(lmod->bl[cnt]);
01356         while(bloop && ((modr = next_bucket_loop(bloop)))) {
01357             if (!(item = ldap_reqtoarr(modr, cnt))) {
01358                 ldap_mods_free(modarr, 1);
01359                 objunref(ld);
01360                 return LDAP_NO_MEMORY;
01361             }
01362             *tmp = item;
01363             tmp++;
01364             objunref(modr);
01365         }
01366         objunref(bloop);
01367     }
01368     *tmp = NULL;
01369
01370     objlock(ld);
01371     res = ldap_modify_ext_s(ld->ldap, lmod->dn, modarr, ld->sctrlsp, NULL);
01372     objunlock(ld);
01373     ldap_mods_free(modarr, 1);
01374     objunref(ld);
01375     return res;
01376 }
```

12.22.4.6 const char* ldap_errmsg (int *res*)

Return LDAP error for a ldap error.

Parameters

res	LDAP error id.

Returns

Error string.

Definition at line 576 of file openldap.c.

```
00576                                       {
00577     return ldap_err2string(res);
00578 }
```

12.22.4.7 struct ldap_attr* ldap_getattr (struct ldap_entry * *entry*, const char * *attr*)

Find and return attribute in a entry.

Parameters

entry	Entry to return attribute from.
attr	Atttribute to return.

Returns

Attribute reference matching attr.

Definition at line 1108 of file openldap.c.

References ldap_entry::attrs, and bucket_list_find_key().

```
01108                                                                                {
01109     if (!entry || !entry->attrs) {
01110         return NULL;
01111     }
01112     return (struct ldap_attr *)bucket_list_find_key(entry->
      attrs, attr);
01113 }
```

12.22.4.8 struct ldap_entry* ldap_getentry (struct ldap_results * *results*, const char * *dn*)

Find and return the entry from the results for a specific dn.

Parameters

results	Results to search in.
dn	DN search for.

Returns

Entry for a DN in the results or NULL.

Definition at line 1096 of file openldap.c.

References bucket_list_find_key(), and ldap_results::entries.

```
01096                                                                              {
01097     if (!results || !dn) {
01098         return NULL;
01099     }
01100     return (struct ldap_entry *)bucket_list_find_key(results->
      entries, dn);
01101 }
```

12.22.4.9 int ldap_mod_add (struct ldap_modify * *lmod*, const char * *attr*, ...)

Add values to a attribute.

Parameters

lmod	LDAP modification referenece.
attr	Attribute to modify.
...	Values to add.

Returns

Zero on success.

Definition at line 1238 of file openldap.c.

References objunref().

Referenced by ldap_mod_addattr().

```
01238                                                                                {
01239     va_list a_list;
01240     char *val;
01241     struct ldap_modreq *modr;
01242
01243     if (!(modr = getmodreq(lmod, attr, LDAP_MOD_ADD))) {
01244         return 1;
01245     }
01246
01247     va_start(a_list, attr);
01248     while((val = va_arg(a_list, void *))) {
01249         if (add_modifyval(modr, val)) {
01250             objunref(modr);
01251             return(1);
01252         }
01253     }
01254
01255     objunref(modr);
01256     va_end(a_list);
01257     return 0;
01258 }
```

12.22.4.10 int ldap_mod_addattr (struct ldap_conn * *ldap*, const char * *dn*, const char * *attr*, const char * *value*)

Add a value for a attribute in a DN.

Parameters

ldap	Reference to the connection.
dn	DN to remove values from.
attr	Attribute to add value to.
value	Value to remove from attribute.

Returns

Non zero ldap error on failure

Definition at line 1416 of file openldap.c.

References ldap_domodify(), ldap_mod_add(), ldap_modifyinit(), and objunref().

```
01416                                                                                {
01417     int res = 0;
01418     struct ldap_modify *lmod;
01419
01420     if (!(lmod = ldap_modifyinit(dn))) {
01421         return LDAP_NO_MEMORY;
01422     }
01423
01424     if (ldap_mod_add(lmod, attr, value, NULL)) {
01425         objunref(lmod);
01426         return LDAP_NO_MEMORY;
01427     }
01428
01429     res = ldap_domodify(ldap, lmod);
01430     objunref(lmod);
01431     return res;
01432 }
```

12.22.4.11 int ldap_mod_del (struct ldap_modify * *lmod*, const char * *attr*, ...)

Delete values from a attribute.

Parameters

lmod	LDAP modification referenece.
attr	Attribute to modify.
...	Values to remove.

Returns

Zero on success.

Definition at line 1211 of file openldap.c.

References objunref().

Referenced by ldap_mod_delattr().

```
01211                                                                   {
01212     va_list a_list;
01213     char *val;
01214     struct ldap_modreq *modr;
01215
01216     if (!(modr = getmodreq(lmod, attr, LDAP_MOD_DELETE))) {
01217         return 1;
01218     }
01219
01220     va_start(a_list, attr);
01221     while((val = va_arg(a_list, void *))) {
01222         if (add_modifyval(modr, val)) {
01223             objunref(modr);
01224             return(1);
01225         }
01226     }
01227
01228     objunref(modr);
01229     va_end(a_list);
01230     return 0;
01231 }
```

12.22.4.12 int ldap_mod_delattr (struct ldap_conn * *ldap*, const char * *dn*, const char * *attr*, const char * *value*)

Delete a value from a attribute in a DN.

Parameters

ldap	Reference to the connection.
dn	DN to remove values from.
attr	Attribute to remove values from.
value	Value to remove from attribute.

Returns

Non zero ldap error on failure

Definition at line 1384 of file openldap.c.

References ldap_domodify(), ldap_mod_del(), ldap_modifyinit(), and objunref().

Referenced by ldap_mod_rematttr().

```
01384                                                                   {
01385     struct ldap_modify *lmod;
01386     int res;
01387
01388     if (!(lmod = ldap_modifyinit(dn))) {
01389         return LDAP_NO_MEMORY;
01390     }
01391     if (ldap_mod_del(lmod, attr, value, NULL)) {
01392         objunref(lmod);
01393         return LDAP_NO_MEMORY;
01394     }
01395
01396     res = ldap_domodify(ldap, lmod);
01397     objunref(lmod);
01398     return res;
01399 }
```

12.22.4.13 int ldap_mod_rematttr (struct ldap_conn * *ldap,* const char * *dn,* const char * *attr*)

Delete a attribute from a DN.

Parameters

ldap	Reference to the connection.
dn	DN to remove attribute from.
attr	Attribute to remove.

Returns

Non zero ldap error on failure

Definition at line 1406 of file openldap.c.

References ldap_mod_delattr().

```
01406                                                                                 {
01407     return ldap_mod_delattr(ldap, dn, attr, NULL);
01408 }
```

12.22.4.14 int ldap_mod_rep (struct ldap_modify * *lmod,* const char * *attr,* ...)

Replace a attribute.

Parameters

lmod	LDAP modification referenece.
attr	Attribute to modify.
...	Values to replace.

Returns

Zero on success.

Definition at line 1265 of file openldap.c.

References objunref().

Referenced by ldap_mod_repattr().

```
01265                                                                                 {
01266     va_list a_list;
01267     char *val;
01268     struct ldap_modreq *modr;
01269
01270     if (!(modr = getmodreq(lmod, attr, LDAP_MOD_REPLACE))) {
01271         return 1;
01272     }
01273
01274     va_start(a_list, attr);
01275     while((val = va_arg(a_list, void *))) {
01276         if (add_modifyval(modr, val)) {
01277             objunref(modr);
01278             return(1);
01279         }
01280     }
01281
01282     objunref(modr);
01283     va_end(a_list);
01284     return 0;
01285 }
```

12.22.4.15 int ldap_mod_repattr (struct ldap_conn * *ldap,* const char * *dn,* const char * *attr,* const char * *value*)

Replace the value of a attribute in a DN.

Parameters

ldap	Reference to the connection.
dn	DN to replace attribute in.
attr	Attribute to replace.
value	Value to replace attr with.

Returns

Non zero ldap error on failure

Definition at line 1441 of file openldap.c.

References ldap_domodify(), ldap_mod_rep(), ldap_modifyinit(), and objunref().

```
01441                                                                                                                {
01442     struct ldap_modify *lmod;
01443     int res;
01444
01445     if (!(lmod = ldap_modifyinit(dn))) {
01446         return LDAP_NO_MEMORY;
01447     }
01448
01449     if (ldap_mod_rep(lmod, attr, value, NULL)) {
01450         objunref(lmod);
01451         return LDAP_NO_MEMORY;
01452     }
01453
01454     res = ldap_domodify(ldap, lmod);
01455     objunref(lmod);
01456     return res;
01457 }
```

12.22.4.16 struct ldap_modify* ldap_modifyinit (const char * *dn*)

Create a modification reference for a DN.

Parameters

dn	DN to modify.

Returns

Reference to a modification structure used to modify a DN.

Definition at line 1118 of file openldap.c.

References ALLOC_CONST, ldap_modify::bl, create_bucketlist(), ldap_modify::dn, objalloc(), and objunref().

Referenced by ldap_mod_addattr(), ldap_mod_delattr(), and ldap_mod_repattr().

```
01118                                                                {
01119     struct ldap_modify *mod;
01120     int cnt;
01121
01122     if (!(mod = objalloc(sizeof(*mod), free_modify))) {
01123         return NULL;
01124     }
01125
01126     ALLOC_CONST(mod->dn, dn);
01127     if (!mod->dn) {
01128         objunref(mod);
01129         return NULL;
01130     }
01131
01132     for(cnt=0; cnt < 3; cnt++) {
01133         if (!(mod->bl[cnt] = create_bucketlist(4, modify_hash))) {
01134             objunref(mod);
01135             return NULL;
01136         }
01137     }
01138
01139     return mod;
01140 }
```

12.22.4.17 int ldap_saslbind (struct ldap_conn * *ld*, const char * *mech*, const char * *realm*, const char * *authcid*, const char * *passwd*, const char * *authzid*)

Bind to the server with SASL.

Parameters

ld	Reference to LDAP connection.
mech	SASL mechanisim.
realm	SASL realm.
authcid	SASL auth id.
passwd	Password for authid.
authzid	Proxy authid.

Returns

-1 on error.

Definition at line 524 of file openldap.c.

References ALLOC_CONST, sasl_defaults::authcid, sasl_defaults::authzid, ldap_conn::ldap, sasl_defaults::mech, objalloc(), objlock(), objref(), objunlock(), objunref(), sasl_defaults::passwd, sasl_defaults::realm, ldap_conn::sasl, and ldap_conn::sctrlsp.

```
00524
                                                {
00525     struct sasl_defaults *sasl;
00526     int res, sasl_flags = LDAP_SASL_AUTOMATIC | LDAP_SASL_QUIET;
00527
00528     if (!objref(ld)) {
00529         return LDAP_UNAVAILABLE;
00530     }
00531
00532     if (!(sasl = objalloc(sizeof(*sasl), free_sasl))) {
00533         return LDAP_NO_MEMORY;
00534     }
00535
00536     ALLOC_CONST(sasl->passwd, passwd);
00537
00538     if (mech) {
00539         ALLOC_CONST(sasl->mech, mech);
00540     } else {
00541         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_MECH, &sasl->mech);
00542     }
00543
00544     if (realm) {
00545         ALLOC_CONST(sasl->realm, realm);
00546     } else {
00547         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_REALM, &sasl->realm );
00548     }
00549
00550     if (authcid) {
00551         ALLOC_CONST(sasl->authcid, authcid);
00552     } else {
00553         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHCID, &sasl->authcid );
00554     }
00555
00556     if (authzid) {
00557         ALLOC_CONST(sasl->authzid, authzid);
00558     } else {
00559         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHZID, &sasl->authzid );
00560     }
00561
00562     objlock(ld);
00563     if (ld->sasl) {
00564         objunref(ld->sasl);
00565     }
00566     ld->sasl = sasl;
00567     res = ldap_sasl_interactive_bind_s(ld->ldap, NULL, sasl->mech, ld->
      sctrlsp , NULL, sasl_flags, dts_sasl_interact, sasl);
00568     objunlock(ld);
00569     objunref(ld);
00570     return res;
00571 }
```

12.22.4.18 struct ldap_results* ldap_search_base (struct ldap_conn * *ld,* const char * *base,* const char * *filter,* int *b64enc,* int * *res,* ...)

Search LDAP connection base.

Parameters

ld	Reference to LDAP connection.
base	Search base dn.
filter	Search filter.
b64enc	Base 64 encode attributes.
res	Pointer containing LDAP error.
...	NULL termincated list of attributes to include.

Returns

Search results structure.

Definition at line 669 of file openldap.c.

```
00669
                                {
00670     va_list a_list;
00671     char *attr, **tmp, **attrs = NULL;
00672     int cnt = 1;
00673
00674     va_start(a_list, res);
00675     while (( attr=va_arg(a_list, void *))) {
00676         cnt++;
00677     }
00678     va_end(a_list);
00679
00680     if (cnt > 1) {
00681         tmp = attrs = malloc(sizeof(void *)*cnt);
00682
00683         va_start(a_list, res);
00684         while (( attr=va_arg(a_list, char *))) {
00685             *tmp = attr;
00686             tmp++;
00687         }
00688         va_end(a_list);
00689         *tmp=NULL;
00690     }
00691
00692     return _dtsldapsearch(ld, base, LDAP_SCOPE_BASE, filter, attrs, b64enc, res);
00693 }
```

12.22.4.19 struct ldap_results* ldap_search_one (struct ldap_conn * *ld*, const char * *base*, const char * *filter*, int *b64enc*, int * *res*, ...)

Search LDAP connection one level.

Parameters

ld	Reference to LDAP connection.
base	Search base dn.
filter	Search filter.
b64enc	Base 64 encode attributes.
res	Pointer containing LDAP error.
...	NULL termincated list of attributes to include.

Returns

Search results structure.

Definition at line 635 of file openldap.c.

```
00635
                                {
00636     va_list a_list;
00637     char *attr, **tmp, **attrs = NULL;
00638     int cnt = 1;
00639
```

```
00640     va_start(a_list, res);
00641     while (( attr=va_arg(a_list, void *))) {
00642         cnt++;
00643     }
00644     va_end(a_list);
00645
00646     if (cnt > 1) {
00647         tmp = attrs = malloc(sizeof(void *)*cnt);
00648
00649         va_start(a_list, res);
00650         while (( attr=va_arg(a_list, char *))) {
00651             *tmp = attr;
00652             tmp++;
00653         }
00654         va_end(a_list);
00655         *tmp=NULL;
00656     }
00657
00658     return _dtsldapsearch(ld, base, LDAP_SCOPE_ONELEVEL, filter, attrs, b64enc, res);
00659 }
```

12.22.4.20 struct ldap_results* ldap_search_sub (struct ldap_conn * *ld*, const char * *base*, const char * *filter*, int *b64enc*, int * *res*, ...)

Search LDAP connection subtree.

Parameters

ld	Reference to LDAP connection.
base	Search base dn.
filter	Search filter.
b64enc	Base 64 encode attributes.
res	Pointer containing LDAP error.
...	NULL termincated list of attributes to include.

Returns

Search results structure.

Definition at line 601 of file openldap.c.

Referenced by ldap_simplerebind().

```
00601
                          {
00602     va_list a_list;
00603     char *attr, **tmp, **attrs = NULL;
00604     int cnt = 1;
00605
00606     va_start(a_list, res);
00607     while (( attr=va_arg(a_list, void *))) {
00608         cnt++;
00609     }
00610     va_end(a_list);
00611
00612     if (cnt > 1) {
00613         tmp = attrs = malloc(sizeof(void *)*cnt);
00614
00615         va_start(a_list, res);
00616         while (( attr=va_arg(a_list, char *))) {
00617             *tmp = attr;
00618             tmp++;
00619         }
00620         va_end(a_list);
00621         *tmp=NULL;
00622     }
00623
00624     return _dtsldapsearch(ld, base, LDAP_SCOPE_SUBTREE, filter, attrs, b64enc, res);
00625 }
```

12.22.4.21 int ldap_simplebind (struct ldap_conn * *ld*, const char * *dn*, const char * *passwd*)

Bind to the connection with simple bind requireing a distingushed name and password.

Parameters

ld	LDAP connection to bind to.
dn	Distinguished name to bind with.
passwd	Password for dn.

Returns

-1 on error.

Definition at line 434 of file openldap.c.

References ldap_simple::cred, ldap_simple::dn, ldap_conn::ldap, objalloc(), objlock(), objref(), objunlock(), objunref(), ldap_conn::sctrlsp, and ldap_conn::simple.

Referenced by ldap_simplerebind().

```
00434                                                                                {
00435     struct ldap_simple *simple;
00436     struct berval *cred;
00437     int res, len = 0;
00438
00439     if (!objref(ld)) {
00440         return LDAP_UNAVAILABLE;
00441     }
00442
00443     if (passwd) {
00444         len = strlen(passwd);
00445     }
00446     simple = objalloc(sizeof(*simple), free_simple);
00447     cred = calloc(sizeof(*cred), 1);
00448     cred->bv_val = malloc(len);
00449     memcpy(cred->bv_val, passwd, len);
00450     cred->bv_len=len;
00451     simple->cred = cred;
00452     simple->dn = strdup(dn);
00453
00454     objlock(ld);
00455     if (ld->simple) {
00456         objunref(ld->simple);
00457     }
00458     ld->simple = simple;
00459     res = ldap_sasl_bind_s(ld->ldap, simple->dn, LDAP_SASL_SIMPLE, simple->
      cred, ld->sctrlsp, NULL, NULL);
00460     objunlock(ld);
00461     objunref(ld);
00462     return res;
00463 }
```

12.22.4.22 int ldap_simplerebind (struct ldap_conn * *ldap*, const char * *initialdn*, const char * *initialpw*, const char * *base*, const char * *filter*, const char * *uidrdn*, const char * *uid*, const char * *passwd*)

Bind to LDAP connection using rebind.

Bind to a connection with a lower privlidge distingushed name and password search for a user dn, bind to the connection with the retrieved dn and user password.

Parameters

ldap	LDAP connection to bind too.
initialdn	Initial dn to bind with.
initialpw	Password for the initial dn.
base	Search base to find user.
filter	LDAP filter to apply to find user.

uidrdn	Attribute containing user id.
uid	To search and bind as.
passwd	Password for the user id.

Returns

-1 on error.

Definition at line 478 of file openldap.c.

References ldap_results::count, ldap_entry::dn, ldap_results::first_entry, ldap_search_sub(), ldap_simplebind(), objref(), and objunref().

```
00479                                                                                {
00480     int res, flen;
00481     struct ldap_results *results;
00482     const char *sfilt;
00483 
00484     if (!objref(ldap)) {
00485         return LDAP_UNAVAILABLE;
00486     }
00487 
00488     if ((res = ldap_simplebind(ldap, initialdn, initialpw))) {
00489         objunref(ldap);
00490         return res;
00491     }
00492 
00493     flen=strlen(uidrdn) + strlen(filter) + strlen(uid) + 7;
00494     sfilt = malloc(flen);
00495     snprintf((char *)sfilt, flen, "(&(%s=%s)%s)", uidrdn, uid, filter);
00496 
00497     if (!(results = ldap_search_sub(ldap, base, sfilt, 0, &res, uidrdn, NULL))) {
00498         free((void *)sfilt);
00499         objunref(ldap);
00500         return res;
00501     }
00502     free((void *)sfilt);
00503 
00504     if (results->count != 1) {
00505         objunref(results);
00506         objunref(ldap);
00507         return LDAP_INAPPROPRIATE_AUTH;
00508     }
00509 
00510     res = ldap_simplebind(ldap, results->first_entry->
    dn, passwd);
00511     objunref(ldap);
00512     objunref(results);
00513     return res;
00514 }
```

12.22.4.23 void ldap_unref_attr (struct ldap_entry * *entry*, struct ldap_attr * *attr*)

Remove a attribute from a entry.

Parameters

entry	The entry to remove attr from.
attr	Attribute to remove.

Definition at line 1059 of file openldap.c.

References ldap_entry::attrs, ldap_entry::first_attr, ldap_attr::next, objcnt(), objunref(), and remove_bucket_item().

```
01059                                                                    {
01060     if (!entry || !attr) {
01061         return;
01062     }
01063 
01064     if (objcnt(attr) > 1) {
01065         objunref(attr);
01066     } else {
01067         if (attr == entry->first_attr) {
01068             entry->first_attr = attr->next;
```

```
01069         }
01070         remove_bucket_item(entry->attrs, attr);
01071     }
01072 }
```

12.22.4.24 void ldap_unref_entry (struct ldap_results * *results*, struct ldap_entry * *entry*)

Remove a entry from a result.

Parameters

results	The result to remove entry from.
entry	Entry to remove.

Definition at line 1077 of file openldap.c.

References ldap_results::entries, ldap_results::first_entry, ldap_entry::next, objcnt(), objunref(), and remove_bucket_item().

```
01077                                                                                 {
01078     if (!results || !entry) {
01079         return;
01080     }
01081
01082     if (objcnt(entry) > 1) {
01083         objunref(entry);
01084     } else {
01085         if (entry == results->first_entry) {
01086             results->first_entry = entry->next;
01087         }
01088         remove_bucket_item(results->entries, entry);
01089     }
01090 }
```

12.23 XML Interface

Utilities for managing XML documents.

Modules

- XSLT Interface

  Utilities for managing XML documents.

Files

- file libxml2.c

  XML Interface.

Data Structures

- struct xml_attr

  XML attribute name value pair.
- struct xml_node

  Reference to a XML Node.
- struct xml_node_iter

  Iterator to traverse nodes in a xpath.
- struct xml_search

  XML xpath search result.

Typedefs

- typedef struct xml_node xml_node

  Forward decleration of structure.
- typedef struct xml_search xml_search

  Forward decleration of structure.
- typedef struct xml_doc xml_doc

  Forward decleration of structure.

Functions

- void xml_free_buffer (void *data)

  Reference destructor for xml_buffer.
- struct xml_doc * xml_loaddoc (const char *docfile, int validate)

  Load a XML file into XML document and return reference.
- struct xml_doc * xml_loadbuf (const uint8_t *buffer, uint32_t len, int validate)

  Load a buffer into XML document returning refereence.
- struct xml_node * xml_getrootnode (struct xml_doc *xmldoc)

  Return reference to the root node.
- struct xml_node * xml_getfirstnode (struct xml_search *xpsearch, void **iter)

  Return reference to the first node optionally creating a iterator.
- struct xml_node * xml_getnextnode (void *iter)

  Return the next node.
- struct bucket_list * xml_getnodes (struct xml_search *xpsearch)

Return reference to bucket list containing nodes.

- struct xml_search * xml_xpath (struct xml_doc *xmldata, const char *xpath, const char *attrkey)

  Return a reference to a xpath search result.
- int xml_nodecount (struct xml_search *xsearch)

  Return the number of nodes in the search path.
- struct xml_node * xml_getnode (struct xml_search *xsearch, const char *key)

  Return a node in the search matching key.
- const char * xml_getattr (struct xml_node *xnode, const char *attr)

  Return value of attribute.
- const char * xml_getrootname (struct xml_doc *xmldoc)

  Return the name of the root node.
- void xml_modify (struct xml_doc *xmldoc, struct xml_node *xnode, const char *value)

  Modify a XML node.
- void xml_setattr (struct xml_doc *xmldoc, struct xml_node *xnode, const char *name, const char *value)

  Modify a XML node attribute.
- void xml_createpath (struct xml_doc *xmldoc, const char *xpath)

  Create a path in XML document.
- void xml_appendnode (struct xml_doc *xmldoc, const char *xpath, struct xml_node *child)

  Append a node to a path.
- struct xml_node * xml_addnode (struct xml_doc *xmldoc, const char *xpath, const char *name, const char *value, const char *attrkey, const char *keyval)

  Append a node to a path.
- void xml_unlink (struct xml_node *xnode)

  Unlink a node from the document.
- void xml_delete (struct xml_node *xnode)

  Delete a node from document it is not unrefd and should be.
- char * xml_getbuffer (void *buffer)

  Return the buffer of a xml_buffer structure.
- void * xml_doctobuffer (struct xml_doc *xmldoc)

  Return a dump of a XML document.
- void xml_init ()

  Initialise/Reference the XML library.
- void xml_close ()

  Unreference the XML library.
- void xml_savefile (struct xml_doc *xmldoc, const char *file, int format, int compress)

  Save XML document to a file.

12.23.1 Detailed Description

Utilities for managing XML documents.

12.23.2 Typedef Documentation

12.23.2.1 typedef struct xml_doc xml_doc

Forward decleration of structure.

Definition at line 631 of file dtsapp.h.

12.23.2.2 typedef struct xml_node xml_node

Forward decleration of structure.

Definition at line 625 of file dtsapp.h.

12.23.2.3 typedef struct xml_search xml_search

Forward decleration of structure.

Definition at line 628 of file dtsapp.h.

12.23.3 Function Documentation

12.23.3.1 struct xml_node* xml_addnode (struct xml_doc * *xmldoc,* const char * *xpath,* const char * *name,* const char * *value,* const char * *attrkey,* const char * *keyval*)

Append a node to a path.

Parameters

xmldoc	Reference to XML document.
xpath	Path to add the node too.
name	Node name.
value	Node value.
attrkey	Attribute to create on node.
keyval	Attribute value of attrkey.

Returns

reference to new node.

Definition at line 651 of file libxml2.c.

References objlock(), objref(), objunlock(), and objunref().

Referenced by xml_createpath().

```
00652                                                                   {
00653     struct xml_node *newnode;
00654     xmlNodePtr parent;
00655     xmlNodePtr child;
00656     xmlChar *encval;
00657 
00658     if (!objref(xmldoc)) {
00659         return NULL;
00660     }
00661 
00662     objlock(xmldoc);
00663     if (!(parent = xml_getparent(xmldoc, xpath))) {
00664         objunlock(xmldoc);
00665         objunref(xmldoc);
00666         return NULL;
00667     }
00668 
00669     encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
00670     child = xmlNewDocNode(xmldoc->doc, NULL, (const xmlChar *)name, encval);
00671     xmlFree(encval);
00672     xmlAddChild(parent,child);
00673 
00674     if (attrkey && keyval) {
00675         encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)keyval);
00676         xmlSetProp(child, (const xmlChar *)attrkey, (const xmlChar *)encval);
00677         xmlFree(encval);
00678     }
00679     objunlock(xmldoc);
00680 
00681     if (!(newnode = xml_nodetohash(xmldoc, child, attrkey))) {
00682         objunref(xmldoc);
00683         return NULL;
```

```
00684     }
00685
00686     objunref(xmldoc);
00687
00688     return newnode;
00689 }
```

12.23.3.2 void xml_appendnode (struct xml_doc * *xmldoc,* const char * *xpath,* struct xml_node * *child*)

Append a node to a path.

Note

The child will most likely be a node unlinked and moved.

Parameters

xmldoc	Reference to XML document.
xpath	Path to add the node too.
child	XML node to append to path.

Definition at line 625 of file libxml2.c.

References xml_node::nodeptr, objlock(), objref(), objunlock(), and objunref().

```
00625                                                                                                                {
00626     xmlNodePtr parent;
00627
00628     if (!objref(xmldoc)) {
00629         return;
00630     }
00631
00632     objlock(xmldoc);
00633     if (!(parent = xml_getparent(xmldoc, xpath))) {
00634         objunlock(xmldoc);
00635         objunref(xmldoc);
00636     }
00637
00638     xmlAddChild(parent,child->nodeptr);
00639     objunlock(xmldoc);
00640     objunref(xmldoc);
00641 }
```

12.23.3.3 void xml_close ( )

Unreference the XML library.

Ideally this should be done after a call to xml_init at shutdown.

Definition at line 758 of file libxml2.c.

References objunref().

```
00758                         {
00759     if (xml_has_init_parser) {
00760         objunref(xml_has_init_parser);
00761     }
00762 }
```

12.23.3.4 void xml_createpath (struct xml_doc * *xmldoc,* const char * *xpath*)

Create a path in XML document.

Note

xpath is not a full xpath just a path [no filters].

Parameters

xmldoc	Reference to XML document.
xpath	Path to create.

Definition at line 507 of file libxml2.c.

References xml_node::name, objlock(), objref(), objunlock(), objunref(), and xml_addnode().

```
00507                                                                   {
00508     struct xml_node *nn;
00509     xmlXPathObjectPtr xpathObj;
00510     char *lpath, *tok, *save, *cpath, *dup;
00511     const char *root = (char *)xmldoc->root->name;
00512     int len;
00513 
00514 
00515     if (!objref(xmldoc)) {
00516         return;
00517     }
00518 
00519     if (!(dup = strdup(xpath))) {
00520         objunref(xmldoc);
00521         return;
00522     }
00523 
00524     len = strlen(xpath)+1;
00525     if (!(cpath = malloc(len))) {
00526         free(dup);
00527         objunref(xmldoc);
00528         return;
00529     }
00530     if (!(lpath = malloc(len))) {
00531         free(dup);
00532         free(cpath);
00533         objunref(xmldoc);
00534         return;
00535     }
00536 
00537     cpath[0] = '\0';
00538     lpath[0] = '\0';
00539 
00540 #ifndef __WIN32__
00541         for (tok = strtok_r(dup, "/", &save); tok ; tok = strtok_r(NULL, "/", &save)) {
00542 #else
00543         for (tok = strtok_s(dup, "/", &save); tok ; tok = strtok_s(NULL, "/", &save)) {
00544 #endif
00545         strcat(cpath,"/");
00546         strcat(cpath, tok);
00547         if (!strcmp(tok, root)) {
00548             strcat(lpath,"/");
00549             strcat(lpath, tok);
00550             continue;
00551         }
00552 
00553         objlock(xmldoc);
00554         if (!(xpathObj = xmlXPathEvalExpression((const xmlChar *)cpath, xmldoc->xpathCtx))) {
00555             objunlock(xmldoc);
00556             free(lpath);
00557             free(cpath);
00558             free(dup);
00559             objunref(xmldoc);
00560             return;
00561         }
00562         objunlock(xmldoc);
00563 
00564         if (xmlXPathNodeSetIsEmpty(xpathObj->nodesetval)) {
00565             nn = xml_addnode(xmldoc, lpath, tok, NULL, NULL, NULL);
00566             objunref(nn);
00567         }
00568 
00569         xmlXPathFreeObject(xpathObj);
00570         strcat(lpath,"/");
00571         strcat(lpath, tok);
00572     }
00573 
00574     free(dup);
00575     free(lpath);
00576     free(cpath);
00577     objunref(xmldoc);
00578 }
```

12.23.3.5 void xml_delete (struct xml_node * *xnode*)

Delete a node from document it is not unrefd and should be.

Parameters

xnode	Reference to node to delete this must be unreferenced after calling this function.

Definition at line 701 of file libxml2.c.

References xml_node::nodeptr, objlock(), and objunlock().

```
00701                                                {
00702     objlock(xnode);
00703     xmlUnlinkNode(xnode->nodeptr);
00704     xmlFreeNode(xnode->nodeptr);
00705     xnode->nodeptr = NULL;
00706     objunlock(xnode);
00707 }
```

12.23.3.6 void* xml_doctobuffer (struct xml_doc * *xmldoc*)

Return a dump of a XML document.

The result can be acessed using xml_getbuffer()

Parameters

xmldoc	Reference to a XML document.

Returns

Reference to a xml_buffer structure.

Definition at line 726 of file libxml2.c.

References objalloc(), objlock(), objunlock(), and xml_free_buffer().

```
00726                                                        {
00727     struct xml_buffer *xmlbuf;
00728
00729     if (!(xmlbuf = objalloc(sizeof(*xmlbuf),xml_free_buffer))) {
00730         return NULL;
00731     }
00732
00733     objlock(xmldoc);
00734     xmlDocDumpFormatMemory(xmldoc->doc, &xmlbuf->buffer, &xmlbuf->size, 1);
00735     objunlock(xmldoc);
00736     return xmlbuf;
00737 }
```

12.23.3.7 void xml_free_buffer (void * *data*)

Reference destructor for xml_buffer.

Warning

do not call this directly.

Definition at line 46 of file libxml2.c.

Referenced by xml_doctobuffer(), and xslt_apply_buffer().

```
00046                                   {
00047     struct xml_buffer *xb = data;
00048     xmlFree(xb->buffer);
00049 }
```

12.23.3.8 const char* xml_getattr (struct xml_node * *xnode*, const char * *attr*)

Return value of attribute.

Parameters

xnode	XML node reference.
attr	Attribute to search for.

Returns

Value of the attribute valid while reference to node is held.

Definition at line 440 of file libxml2.c.

References xml_node::attrs, bucket_list_find_key(), objunref(), and xml_attr::value.

```
00440                                                                   {
00441     struct xml_attr *ainfo;
00442
00443     if (!xnode) {
00444         return NULL;
00445     }
00446
00447     if ((ainfo = bucket_list_find_key(xnode->attrs, attr))) {
00448         objunref(ainfo);
00449         return ainfo->value;
00450     } else {
00451         return NULL;
00452     }
00453 }
```

12.23.3.9 char* xml_getbuffer (void * *buffer*)

Return the buffer of a xml_buffer structure.

Note

only valid while reference is held to the xml_buffer struct.

Parameters

buffer	Reference to a xml_buffer struct.

Definition at line 712 of file libxml2.c.

```
00712                                       {
00713     struct xml_buffer *xb = buffer;
00714
00715     if (!xb) {
00716         return NULL;
00717     }
00718     return (char *)xb->buffer;
00719 }
```

12.23.3.10 struct xml_node* xml_getfirstnode (struct xml_search * *xpsearch*, void ** *iter*)

Return reference to the first node optionally creating a iterator.

Setting the optional iterator and using it on future calls to xml_getnextnode its possible to iterate through the search path.

Todo Thread safety when XML doc changes.

Note

using xml_getnodes() returns a bucket list of nodes this is prefered.

Warning

This is not thread safe.

Parameters

xpsearch	XML xpath search to find first node.
iter	Optional iterator created and returned (must be unreferenced)

Returns

Reference to first node in the path.

Definition at line 295 of file libxml2.c.

References xml_node_iter::cnt, xml_node_iter::curpos, objalloc(), objlock(), objref(), objunlock(), objunref(), xml_nodecount(), and xml_node_iter::xsearch.

```
00295                                                                    {
00296     struct xml_node_iter *newiter;
00297     struct xml_node *xn;
00298 
00299     if (!objref(xpsearch)) {
00300         return NULL;
00301     }
00302 
00303     if (iter) {
00304         newiter = objalloc(sizeof(*newiter), free_iter);
00305         objlock(xpsearch);
00306         newiter->cnt = xml_nodecount(xpsearch);
00307         objunlock(xpsearch);
00308         newiter->curpos = 0;
00309         newiter->xsearch = xpsearch;
00310         objref(newiter->xsearch);
00311         *iter = newiter;
00312     }
00313 
00314     xn = xml_gethash(xpsearch, 0, NULL);
00315     objunref(xpsearch);
00316     return xn;
00317 }
```

12.23.3.11 struct xml_node* xml_getnextnode (void * *iter*)

Return the next node.

Parameters

iter	Iterator set in call to from xml_getfirstnode.

Returns

Reference to next node.

Definition at line 322 of file libxml2.c.

References xml_node_iter::cnt, xml_node_iter::curpos, objlock(), objref(), objunlock(), objunref(), and xml_node_iter::xsearch.

```
00322                                                {
00323     struct xml_node_iter *xi = iter;
00324     struct xml_node *xn;
00325 
00326     if (!objref(xi->xsearch)) {
00327         return NULL;
00328     }
00329 
00330     objlock(xi);
00331     xi->curpos ++;
00332     if (xi->curpos >= xi->cnt) {
00333         objunlock(xi);
00334         objunref(xi->xsearch);
00335         return NULL;
00336     }
00337     xn = xml_gethash(xi->xsearch, xi->curpos, NULL);
00338     objunlock(xi);
```

```
00339     objunref(xi->xsearch);
00340
00341     return xn;
00342 }
```

12.23.3.12 struct xml_node* xml_getnode (struct xml_search * *xsearch*, const char * *key*)

Return a node in the search matching key.

The key is matched against the index attribute supplied or the value of the node.

Parameters

xsearch	Reference to xpath search.
key	Value to use to find node matched aginst the index attribute/value.

Returns

Reference to XML node.

Definition at line 429 of file libxml2.c.

References bucket_list_find_key(), and xml_search::nodes.

```
00429                                                                    {
00430     if (!xsearch) {
00431         return NULL;
00432     }
00433     return bucket_list_find_key(xsearch->nodes, key);
00434 }
```

12.23.3.13 struct bucket_list* xml_getnodes (struct xml_search * *xpsearch*)

Return reference to bucket list containing nodes.

Note

use of this is prefered to xml_getfirstnode() / xml_getnextnode() if search order is not a issue.

Parameters

xpsearch	Reference to xpath search result returned by xml_xpath.

Returns

Reference to bucket list containing nodes.

Definition at line 349 of file libxml2.c.

References xml_search::nodes, and objref().

```
00349                                                                    {
00350     return (xpsearch && objref(xpsearch->nodes)) ? xpsearch->nodes : NULL;
00351 }
```

12.23.3.14 const char* xml_getrootname (struct xml_doc * *xmldoc*)

Return the name of the root node.

Note

do not free or unref this.

Parameters

xmldoc	XML Document.

Definition at line 458 of file libxml2.c.

```
00458                                                       {
00459     if (xmldoc) {
00460         return (const char *)xmldoc->root->name;
00461     }
00462     return NULL;
00463 }
```

12.23.3.15 struct xml_node* xml_getrootnode (struct xml_doc * *xmldoc*)

Return reference to the root node.

Parameters

xmldoc	XML Document to find root in.

Definition at line 276 of file libxml2.c.

References objlock(), and objunlock().

```
00276                                                           {
00277     struct xml_node *rn;
00278
00279     objlock(xmldoc);
00280     rn = xml_nodetohash(xmldoc, xmldoc->root, NULL);
00281     objunlock(xmldoc);
00282     return rn;
00283 }
```

12.23.3.16 void xml_init ( )

Initialise/Reference the XML library.

Ideally this should be done on application startup but will be started and stoped as needed.

Definition at line 742 of file libxml2.c.

References objalloc(), and objref().

Referenced by xml_loadbuf(), and xml_loaddoc().

```
00742                       {
00743     if (!xml_has_init_parser) {
00744         xml_has_init_parser = objalloc(0, free_parser);
00745         xmlInitParser();
00746         LIBXML_TEST_VERSION
00747         xmlKeepBlanksDefault(0);
00748         xmlLoadExtDtdDefaultValue = 1;
00749         xmlSubstituteEntitiesDefault(1);
00750     } else {
00751         objref(xml_has_init_parser);
00752     }
00753 }
```

12.23.3.17 struct xml_doc* xml_loadbuf (const uint8_t * *buffer*, uint32_t *len*, int *validate*)

Load a buffer into XML document returning refereence.

Parameters

buffer	Buffer containing the XML.
len	Size of the buffer.
validate	Set to non zero value to fail if validation fails.

Returns

XML Document or NULL on failure

Definition at line 168 of file libxml2.c.

References objalloc(), objunref(), and xml_init().

Referenced by curl_buf2xml().

```
00168                                                                                {
00169     struct xml_doc *xmldata;
00170     int flags;
00171
00172     xml_init();
00173
00174     if (!(xmldata = objalloc(sizeof(*xmldata), free_xmldata))) {
00175         return NULL;
00176     }
00177
00178     if (validate) {
00179         flags = XML_PARSE_DTDLOAD | XML_PARSE_DTDVALID;
00180     } else {
00181         flags = XML_PARSE_DTDVALID;
00182     }
00183
00184     if (!(xmldata->doc = xmlReadMemory((const char *)buffer, len, NULL, NULL, flags))) {
00185         objunref(xmldata);
00186         return NULL;
00187     }
00188     return xml_setup_parse(xmldata, 0);
00189 }
```

12.23.3.18 struct xml_doc* xml_loaddoc (const char * *docfile*, int *validate*)

Load a XML file into XML document and return reference.

Parameters

docfile	Pathname to XML file.
validate	Set to non zero value to fail if validation fails.

Returns

XML Document or NULL on failure

Definition at line 146 of file libxml2.c.

References objalloc(), objunref(), and xml_init().

```
00146                                                                        {
00147     struct xml_doc *xmldata;
00148
00149     xml_init();
00150
00151     if (!(xmldata = objalloc(sizeof(*xmldata), free_xmldata))) {
00152         return NULL;
00153     }
00154
00155     if (!(xmldata->doc = xmlParseFile(docfile))) {
00156         objunref(xmldata);
00157         return NULL;
00158     }
00159
00160     return xml_setup_parse(xmldata, validate);
00161 }
```

12.23.3.19 void xml_modify (struct xml_doc * *xmldoc,* struct xml_node * *xnode,* const char * *value*)

Modify a XML node.

Parameters

xmldoc	XML Document node belongs to
xnode	XML Node to modify.
value	Value to set.

Definition at line 469 of file libxml2.c.

References ALLOC_CONST, xml_node::nodeptr, objlock(), objunlock(), and xml_node::value.

```
00469                                                                                {
00470     xmlChar *encval;
00471     xmlNodePtr node;
00472
00473     objlock(xmldoc);
00474     node = xnode->nodeptr;
00475     encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
00476     xmlNodeSetContent(node, encval);
00477     xmlFree(encval);
00478     encval = xmlNodeListGetString(xmldoc->doc, node->xmlChildrenNode, 1);
00479     objunlock(xmldoc);
00480
00481     if (xnode->value) {
00482         free((void*)xnode->value);
00483     }
00484     ALLOC_CONST(xnode->value, (const char *)encval);
00485     xmlFree(encval);
00486 }
```

12.23.3.20 int xml_nodecount (struct xml_search * *xsearch*)

Return the number of nodes in the search path.

Parameters

xsearch	Reference to XML xpath search (xml_xpath())

Returns

Number of of nodes.

Definition at line 413 of file libxml2.c.

References xml_search::xpathObj.

Referenced by xml_getfirstnode().

```
00413                                                  {
00414     xmlNodeSetPtr nodeset;
00415
00416     if (xsearch && xsearch->xpathObj && ((nodeset = xsearch->xpathObj->nodesetval))) {
00417         return nodeset->nodeNr;
00418     } else {
00419         return 0;
00420     }
00421 }
```

12.23.3.21 void xml_savefile (struct xml_doc * *xmldoc*, const char * *file*, int *format*, int *compress*)

Save XML document to a file.

Parameters

xmldoc	Reference to XML document to save.
file	Filename to write the XML document too.
format	Formating flag from libxml2.
compress	Compression level 0[none]-9.

Definition at line 769 of file libxml2.c.

References objlock(), and objunlock().

```
00769                                                                                 {
00770     objlock(xmldoc);
00771     xmlSetDocCompressMode(xmldoc->doc, compress);
00772     xmlSaveFormatFile(file, xmldoc->doc, format);
00773     xmlSetDocCompressMode(xmldoc->doc, 0);
00774     objunlock(xmldoc);
00775 }
```

12.23.3.22 void xml_setattr (struct xml_doc * *xmldoc*, struct xml_node * *xnode*, const char * *name*, const char * *value*)

Modify a XML node attribute.

Parameters

xmldoc	XML Document node belongs to
xnode	XML Node to modify.
name	Attribute to modify.
value	Value to set.

Definition at line 493 of file libxml2.c.

References xml_node::nodeptr, objlock(), and objunlock().

```
00493
        {
00494     xmlChar *encval;
00495
00496     objlock(xmldoc);
00497     encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
00498     xmlSetProp(xnode->nodeptr, (const xmlChar *)name, (const xmlChar *)encval);
00499     objunlock(xmldoc);
00500     xmlFree(encval);
00501 }
```

12.23.3.23 void xml_unlink (struct xml_node * *xnode*)

Unlink a node from the document.

Parameters

xnode	Reference of node to unlink.

Definition at line 693 of file libxml2.c.

References xml_node::nodeptr, objlock(), and objunlock().

```
00693                                                 {
00694     objlock(xnode);
00695     xmlUnlinkNode(xnode->nodeptr);
00696     objunlock(xnode);
00697 }
```

12.23.3.24 struct xml_search* xml_xpath (struct xml_doc * *xmldata*, const char * *xpath*, const char * *attrkey*)

Return a reference to a xpath search result.

Parameters

xmldata	XML Document to search.
xpath	Xpath search to apply.
attrkey	Attribute to index by.

Returns

Reference to XML search result.

Definition at line 381 of file libxml2.c.

References xml_search::nodes, objalloc(), objlock(), objref(), objunlock(), objunref(), xml_search::xmldoc, and xml_search::xpathObj.

```
00381                                                                                                                {
00382     struct xml_search *xpsearch;
00383
00384     if (!objref(xmldata) || !(xpsearch = objalloc(sizeof(*xpsearch), free_xmlsearch))) {
00385         return NULL;
00386     }
00387
00388     objlock(xmldata);
00389     xpsearch->xmldoc = xmldata;
00390     if (!(xpsearch->xpathObj = xmlXPathEvalExpression((const xmlChar *)xpath, xmldata->xpathCtx)))
     {
00391         objunlock(xmldata);
00392         objunref(xpsearch);
00393         return NULL;
00394     }
00395
00396     if (xmlXPathNodeSetIsEmpty(xpsearch->xpathObj->nodesetval)) {
00397         objunlock(xmldata);
00398         objunref(xpsearch);
00399         return NULL;
00400     }
00401     objunlock(xmldata);
00402
00403     if (!(xpsearch->nodes = xml_setnodes(xpsearch, attrkey))) {
00404         objunref(xpsearch);
00405         return NULL;
00406     }
00407     return xpsearch;
00408 }
```

12.24 XSLT Interface

Utilities for managing XML documents.

Files

- file libxslt.c

  XSLT Interface.

Data Structures

- struct xslt_doc

  XSLT Document.
- struct xslt_param

  XSLT Parameter name/value pair.

Typedefs

- typedef struct xslt_doc xslt_doc

  Forward decleration of structure.

Functions

- struct xslt_doc * xslt_open (const char *xsltfile)

  Open a XSLT file returning reference to it.
- void xslt_addparam (struct xslt_doc *xsltdoc, const char *param, const char *value)

  Add a parameter to the XSLT document.
- void xslt_clearparam (struct xslt_doc *xsltdoc)

  Delete all parameters of a XSLT document.
- void xslt_apply (struct xml_doc *xmldoc, struct xslt_doc *xsltdoc, const char *filename, int comp)

  Apply XSLT document to a XML document.
- void * xslt_apply_buffer (struct xml_doc *xmldoc, struct xslt_doc *xsltdoc)

  Apply XSLT document to a XML document returning result in buffer.
- void xslt_init ()

  Reference the XSLT parser.
- void xslt_close ()

  Release reference to XSLT parser.

12.24.1 Detailed Description

Utilities for managing XML documents.

12.24.2 Typedef Documentation

12.24.2.1 typedef struct xslt_doc xslt_doc

Forward decleration of structure.

Definition at line 634 of file dtsapp.h.

12.24.3 Function Documentation

12.24.3.1 void xslt_addparam (struct xslt_doc * *xsltdoc,* const char * *param,* const char * *value*)

Add a parameter to the XSLT document.

Parameters

xsltdoc	Reference to XSLT document.
param	Name of parameter.
value	Parameter value.

Definition at line 94 of file libxslt.c.

References addtobucket(), ALLOC_CONST, xslt_param::name, objalloc(), objlock(), objref(), objunlock(), objunref(), xslt_doc::params, and xslt_param::value.

```
00094                                                                                                {
00095     struct xslt_param *xparam;
00096     int size;
00097
00098     if (!xsltdoc || !xsltdoc->params || !objref(xsltdoc) || !(xparam =
      objalloc(sizeof(*xparam), free_param))) {
00099         return;
00100     }
00101
00102     size = strlen(value) + 3;
00103     ALLOC_CONST(xparam->name, param);
00104     xparam->value = malloc(size);
00105     snprintf((char *)xparam->value, size, "'%s'", value);
00106     objlock(xsltdoc);
00107     addtobucket(xsltdoc->params, xparam);
00108     objunlock(xsltdoc);
00109     objunref(xparam);
00110     objunref(xsltdoc);
00111 }
```

12.24.3.2 void xslt_apply (struct xml_doc * *xmldoc*, struct xslt_doc * *xsltdoc*, const char * *filename*, int *comp*)

Apply XSLT document to a XML document.

Parameters

xmldoc	Reference to XML document.
xsltdoc	Reference to XSLT document.
filename	File to write the result too.
comp	Compression level 0-9 [0 = none].

Definition at line 167 of file libxslt.c.

References xslt_doc::doc, objlock(), objunlock(), objunref(), touch(), and xslt_clearparam().

```
00167                                                                                                {
00168     const char **params = NULL;
00169     xmlDocPtr res;
00170
00171     /* ref's xml/xslt locks xslt IF set*/
00172     if (!(params = xslt_params(xmldoc, xsltdoc))) {
00173         return;
00174     }
00175
00176 #ifndef __WIN32__
00177     touch(filename, 80, 80);
00178 #else
00179     touch(filename);
00180 #endif
00181     objlock(xmldoc);
00182     res = xsltApplyStylesheet(xsltdoc->doc, xmldoc->doc, params);
00183     xsltSaveResultToFilename(filename, res, xsltdoc->doc, comp);
00184     objunlock(xmldoc);
00185     objunref(xmldoc);
00186     objunlock(xsltdoc);
00187
00188     free(params);
00189     xmlFreeDoc(res);
00190     xslt_clearparam(xsltdoc);
00191     objunref(xsltdoc);
00192 }
```

12.24.3.3 void* xslt_apply_buffer (struct xml_doc * *xmldoc,* struct xslt_doc * *xsltdoc*)

Apply XSLT document to a XML document returning result in buffer.

Parameters

xmldoc	Reference to XML document.
xsltdoc	Reference to XSLT document.

Returns

Reference to xml_buffer containing the result of the transform.

Definition at line 198 of file libxslt.c.

References xslt_doc::doc, objalloc(), objlock(), objunlock(), objunref(), xml_free_buffer(), and xslt_clearparam().

```
00198                                                                           {
00199     struct xml_buffer *xmlbuf;
00200     const char **params;
00201     xmlDocPtr res;
00202 
00203     if (!(xmlbuf = objalloc(sizeof(*xmlbuf),xml_free_buffer))) {
00204         return NULL;
00205     }
00206 
00207     if (!(params = xslt_params(xmldoc, xsltdoc))) {
00208         objunref(xmlbuf);
00209         return NULL;
00210     }
00211 
00212     objlock(xmldoc);
00213     res = xsltApplyStylesheet(xsltdoc->doc, xmldoc->doc, params);
00214     xsltSaveResultToString(&xmlbuf->buffer, &xmlbuf->size, res, xsltdoc->doc);
00215     objunlock(xmldoc);
00216     objunref(xmldoc);
00217     objunlock(xsltdoc);
00218 
00219     free(params);
00220     xmlFreeDoc(res);
00221     xslt_clearparam(xsltdoc);
00222     objunref(xsltdoc);
00223 
00224     return xmlbuf;
00225 }
```

12.24.3.4 void xslt_clearparam (struct xslt_doc * *xsltdoc*)

Delete all parameters of a XSLT document.

Parameters

xsltdoc	Reference to XSLT document.

Definition at line 115 of file libxslt.c.

References create_bucketlist(), objlock(), objunlock(), objunref(), and xslt_doc::params.

Referenced by xslt_apply(), and xslt_apply_buffer().

```
00115                                                {
00116     if (!xsltdoc || !xsltdoc->params) {
00117         return;
00118     }
00119 
00120     objlock(xsltdoc);
00121     objunref(xsltdoc->params);
00122     xsltdoc->params = create_bucketlist(0, xslt_hash);
00123     objunlock(xsltdoc);
00124 }
```

12.24.3.5 void xslt_close ( )

Release reference to XSLT parser.

Note

It is best if the application keeps a reference to the parser before use of XSLT and release it on termination.

Definition at line 241 of file libxslt.c.

References objunref().

```
00241                           {
00242     if (xslt_has_init_parser) {
00243         objunref(xslt_has_init_parser);
00244     }
00245 }
```

12.24.3.6 void xslt_init ( )

Reference the XSLT parser.

Note

It is best if the application keeps a reference to the parser before use of XSLT and release it on termination.

Definition at line 230 of file libxslt.c.

References objalloc(), and objref().

Referenced by xslt_open().

```
00230                           {
00231     if (!xslt_has_init_parser) {
00232         xslt_has_init_parser=objalloc(0, free_parser);
00233     } else {
00234         objref(xslt_has_init_parser);
00235     }
00236 }
```

12.24.3.7 struct xslt_doc* xslt_open (const char * *xsltfile*)

Open a XSLT file returning reference to it.

Parameters

xsltfile	XSLT pathname to open.

Returns

Reference to XSLT document.

Definition at line 67 of file libxslt.c.

References create_bucketlist(), xslt_doc::doc, objalloc(), xslt_doc::params, and xslt_init().

```
00067                                                   {
00068     struct xslt_doc *xsltdoc;
00069
00070     if (!(xsltdoc = objalloc(sizeof(*xsltdoc), free_xsltdoc))) {
00071         return NULL;
00072     }
00073     xslt_init();
00074
00075     xsltdoc->doc = xsltParseStylesheetFile((const xmlChar *)xsltfile);
00076     xsltdoc->params = create_bucketlist(0, xslt_hash);
00077     return xsltdoc;
00078 }
```

12.25 CURL Url interface.

Interface to libCURL.

Files

- file curl.c

  CURL Interface.

Data Structures

- struct **curl_progress**

  Allow progress monitoring.
- struct **curl_password**

  CURL Authentification callback.
- struct curl_post

  HTTP post data structure.
- struct basic_auth

  Basic authentification structure.
- struct curlbuf

  Buffer containing the result of a curl transaction.

Typedefs

- typedef struct curl_post curl_post

  Forward decleration of structure.
- typedef struct basic_auth *(* curl_authcb)(const char *, const char *, void *)

  Callback to set the authentification ie on error 401.
- typedef int(* curl_progress_func)(void *, double, double, double, double)

  CURL callback function called when there is progress (CURLOPT_PROGRESSFUNCTION).
- typedef void(* curl_progress_pause)(void *, int)

  Callback function to control the progress bar.
- typedef void *(* curl_progress_newdata)(void *)

  Create a new progress data structure.

Functions

- int curlinit (void)

  Initilise the CURL library.
- void curlclose (void)

  Un reference CURL. This is required for each call to curlinit().
- struct curlbuf * curl_geturl (const char *def_url, struct basic_auth *bauth, curl_authcb authcb, void *auth_data)

  Fetch the URL using CURL (HTTP GET)
- struct curlbuf * curl_posturl (const char *def_url, struct basic_auth *bauth, struct curl_post *post, curl_authcb authcb, void *auth_data)

  Fetch the URL using CURL (HTTP POST)
- struct curlbuf * curl_ungzip (struct curlbuf *cbuf)

  If the buffer contains GZIP data uncompress it.
- struct basic_auth * curl_newauth (const char *user, const char *passwd)

Create a new auth structure with initial vallues.

- struct curl_post * curl_newpost (void)

  Create a HTTP Post data structure.
- void curl_postitem (struct curl_post *post, const char *name, const char *value)

  Add a item value pair to post structure.
- char * url_escape (char *url)

  Escape and return the url.
- char * url_unescape (char *url)

  UN escape and return the url.
- void curl_setprogress (curl_progress_func cb, curl_progress_pause p_cb, curl_progress_newdata d_cb, void *data)

  Configure global progress handling.
- void curl_setauth_cb (curl_authcb auth_cb, void *data)

  Set global password callback.
- struct xml_doc * curl_buf2xml (struct curlbuf *cbuf)

  Create a XML document from from buffer (application/xml)

12.25.1 Detailed Description

Interface to libCURL.

12.25.2 Typedef Documentation

12.25.2.1 typedef struct basic_auth*(* curl_authcb)(const char *, const char *, void *)

Callback to set the authentification ie on error 401.

Parameters

user	Initial username (currently set)
passwd	Initial password (currently set)
data	Reference to data passed.

Returns

New auth structure to re attempt authentification.

Definition at line 853 of file dtsapp.h.

12.25.2.2 typedef struct curl_post curl_post

Forward decleration of structure.

Definition at line 846 of file dtsapp.h.

12.25.2.3 typedef int(* curl_progress_func)(void *, double, double, double, double)

CURL callback function called when there is progress (CURLOPT_PROGRESSFUNCTION).

Parameters

clientp	Reference to userdata supplied.
dltotal	Total download bytes.
dlnow	Current bytes downloaded.
ultotal	Total upload bytes.
ulnow	Current upload bytes.

Returns

Returning a non-zero value from this callback will cause the transfer to abort.

Definition at line 862 of file dtsapp.h.

12.25.2.4 typedef void*(* curl_progress_newdata)(void *)

Create a new progress data structure.

See Also

curl_setprogress()

curl_setprogress() allows setting a default progress callback if set it will call a callback to create a new callback progress userdata for the current session.

Parameters

data	Reference to userdata supplied to curl_setprogress().

Returns

Reference to userdata to be used in current session.

Definition at line 876 of file dtsapp.h.

12.25.2.5 typedef void(* curl_progress_pause)(void *, int)

Callback function to control the progress bar.

Parameters

data	Reference to userdata supplied.
state	one of 0, 1 or -1 for Pause, Unpause and Close respectfully.

Definition at line 867 of file dtsapp.h.

12.25.3 Function Documentation

12.25.3.1 struct xml_doc* curl_buf2xml (struct curlbuf * *cbuf*)

Create a XML document from from buffer (application/xml)

Parameters

cbuf	CURL request buffer.

Returns

Reference to XML document.

Definition at line 489 of file curl.c.

References curlbuf::body, curlbuf::bsize, curlbuf::c_type, curl_ungzip(), and xml_loadbuf().

```
00489                                                       {
00490     struct xml_doc *xmldoc = NULL;
00491
00492     if (cbuf && cbuf->c_type && !strcmp("application/xml", cbuf->c_type)) {
00493         curl_ungzip(cbuf);
00494         xmldoc = xml_loadbuf(cbuf->body, cbuf->bsize, 1);
00495     }
00496     return xmldoc;
00497 }
```

12.25.3.2 struct curlbuf* curl_geturl (const char * *def_url*, struct basic_auth * *bauth*, curl_authcb *authcb*, void * *auth_data*)

Fetch the URL using CURL (HTTP GET)

Note

if no authcb is specified and curl_setauth_cb() has been called this default will be used.

Parameters

def_url	URL to fetch.
bauth	Basic auth structure to initilise auth.
authcb	Callback if authentification is required.
auth_data	Reference to userdata passed in auth callback.

Returns

CURL buffer structure.

Definition at line 276 of file curl.c.

```
00276
        {
00277     return curl_sendurl(def_url, bauth, NULL, authcb, auth_data);
00278 }
```

12.25.3.3 struct basic_auth* curl_newauth (const char * *user*, const char * *passwd*)

Create a new auth structure with initial vallues.

Note

if NULL is supplied its replaced with zero length string

Parameters

user	Optional initial username to set.

passwd	Optional initial password to set.

Returns

Reference to new authentification structure.

Definition at line 328 of file curl.c.

References objalloc(), basic_auth::passwd, and basic_auth::user.

```
00328                                                                    {
00329     struct basic_auth *bauth;
00330
00331     if (!(bauth = (struct basic_auth *)objalloc(sizeof(*bauth), curl_freeauth))) {
00332         return NULL;
00333     }
00334     if (user) {
00335         bauth->user = strdup(user);
00336     } else {
00337         bauth->user = strdup("");
00338     }
00339     if (passwd) {
00340         bauth->passwd = strdup(passwd);
00341     } else {
00342         bauth->passwd = strdup("");
00343     }
00344     return bauth;
00345 }
```

12.25.3.4 struct curl_post* curl_newpost (void)

Create a HTTP Post data structure.

Returns

Reference to new structure.

Definition at line 356 of file curl.c.

References curl_post::first, curl_post::last, and objalloc().

```
00356                                          {
00357     struct curl_post *post;
00358     if (!(post = objalloc(sizeof(*post), free_post))) {
00359         return NULL;
00360     }
00361     post->first = NULL;
00362     post->last = NULL;
00363     return post;
00364 }
```

12.25.3.5 void curl_postitem (struct curl_post * *post*, const char * *name*, const char * *value*)

Add a item value pair to post structure.

Parameters

post	Post structure created with curl_newpost()
name	Name of the pair.
value	Value of the pair.

Definition at line 370 of file curl.c.

References curl_post::first, curl_post::last, objlock(), and objunlock().

```
00370                                                                                                   {
00371     if (!name || !value) {
00372         return;
00373     }
00374     objlock(post);
00375     curl_formadd(&post->first, &post->last,
00376         CURLFORM_COPYNAME, name,
00377         CURLFORM_COPYCONTENTS, value,
00378         CURLFORM_END);
00379     objunlock(post);
00380 }
```

12.25.3.6 struct curlbuf* curl_posturl (const char * *def_url*, struct basic_auth * *bauth*, struct curl_post * *post*, curl_authcb *authcb*, void * *auth_data*)

Fetch the URL using CURL (HTTP POST)

Note

if no authcb is specified and curl_setauth_cb() has been called this default will be used.

Parameters

def_url	URL to fetch.
bauth	Basic auth structure to initilise auth.
post	Reference to curl post structure.
authcb	Callback if authentification is required.
auth_data	Reference to userdata passed in auth callback.

Returns

CURL buffer structure.

Definition at line 288 of file curl.c.

```
00288
                                    {
00289     return curl_sendurl(def_url, bauth, post, authcb, auth_data);
00290 }
```

12.25.3.7 void curl_setauth_cb (curl_authcb *auth_cb*, void * *data*)

Set global password callback.

Note

This will only persist as long as a reference to CURL is held use curlinit() and curlclose() at application startup and shutdown.

Parameters

auth_cb	Authentification call back.
data	Reference to userdata passed in callback.

Definition at line 470 of file curl.c.

References objalloc(), objref(), and objunref().

```
00470                                                   {
00471     if (curlpassword) {
00472         objunref(curlpassword);
00473         curlpassword = NULL;
00474     }
```

```
00475
00476     if (!(curlpassword = objalloc(sizeof(*curlpassword), free_curlpassword))) {
00477         return;
00478     }
00479
00480     curlpassword->authcb = auth_cb;
00481     if (data && objref(data)) {
00482         curlpassword->data = data;
00483     }
00484 }
```

12.25.3.8 void curl_setprogress (curl_progress_func *cb*, curl_progress_pause *p_cb*, curl_progress_newdata *d_cb*, void * *data*)

Configure global progress handling.

Note

This will only persist as long as a reference to CURL is held use curlinit() and curlclose() at application startup and shutdown.

Parameters

cb	CURL progress function callback.
p_cb	CURL progress control (pause) callback.
d_cb	CURL progress data allocation callback.
data	initial data passed to d_cb.

See Also

curl_progress_func()
curl_progress_pause()
curl_progress_newdata()

Definition at line 442 of file curl.c.

References objalloc(), objref(), and objunref().

```
00442
          {
00443     if (curlprogress) {
00444         objunref(curlprogress);
00445         curlprogress = NULL;
00446     }
00447
00448     if (!(curlprogress = objalloc(sizeof(*curlprogress), free_progress))) {
00449         return;
00450     }
00451     curlprogress->cb = cb;
00452     curlprogress->d_cb = d_cb;
00453     curlprogress->p_cb = p_cb;
00454     if (data && objref(data)) {
00455         curlprogress->data = data;
00456     }
00457 }
```

12.25.3.9 struct curlbuf* curl_ungzip (struct curlbuf * *cbuf*)

If the buffer contains GZIP data uncompress it.

Parameters

cbuf	Curl buffer to uncompress.

Returns

Pointer to cbuf with the body replaced uncompressed.

Definition at line 295 of file curl.c.

References curlbuf::body, curlbuf::bsize, gzinflatebuf(), and is_gzip().

Referenced by curl_buf2xml().

```
00295                                                   {
00296     uint8_t *gzbuf;
00297     uint32_t len;
00298
00299     if (is_gzip((uint8_t *)cbuf->body, cbuf->bsize) &&
00300             ((gzbuf = gzinflatebuf((uint8_t *)cbuf->body, cbuf->
      bsize, &len)))) {
00301         free(cbuf->body);
00302         cbuf->body = gzbuf;
00303         cbuf->bsize = len;
00304     }
00305     return cbuf;
00306 }
```

12.25.3.10 void curlclose (void)

Un reference CURL. This is required for each call to curlinit().

Definition at line 122 of file curl.c.

References objunref().

```
00122                       {
00123     objunref(curl_isinit);
00124     curl_isinit = NULL;
00125 }
```

12.25.3.11 int curlinit (void)

Initilise the CURL library.

Note

Curl functions will initilize and unreference curl when done it is best the application hold a reference to benifit from caching. curlclose() Must be called if it has been used

Definition at line 92 of file curl.c.

References objalloc(), objlock(), objref(), objunlock(), and objunref().

Referenced by url_escape(), and url_unescape().

```
00092                       {
00093     if (curl_isinit) {
00094         return objref(curl_isinit);
00095     }
00096
00097     if (!(curl_isinit = objalloc(sizeof(void *),curlfree))) {
00098         return 0;
00099     }
00100
00101     objlock(curl_isinit);
00102     if (!(curl = curl_easy_init())) {
00103         objunlock(curl_isinit);
```

```
00104         objunref(curl_isinit);
00105         return 0;
00106     }
00107
00108     curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0);
00109     curl_easy_setopt(curl, CURLOPT_NOSIGNAL, 1);
00110     curl_easy_setopt(curl, CURLOPT_COOKIEFILE, "");
00111
00112     curl_easy_setopt(curl, CURLOPT_USERAGENT, "libcurl-agent/1.0 [Distro Solutions]");
00113
00114     curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, bodytobuffer);
00115     curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, headertobuffer);
00116     objunlock(curl_isinit);
00117     return 1;
00118 }
```

12.25.3.12 char* url_escape (char * *url*)

Escape and return the url.

Parameters

url	URL to escape

Returns

A malloc()'d URL that needs to be free()'d

Definition at line 385 of file curl.c.

References curlinit(), objlock(), objunlock(), and objunref().

```
00385                                   {
00386     char *esc;
00387     char *ret = NULL;
00388
00389     if (!curlinit()) {
00390         return NULL;
00391     }
00392
00393     objlock(curl_isinit);
00394     esc = curl_easy_escape(curl, url, 0);
00395     if (esc) {
00396         ret = strdup(esc);
00397     }
00398     curl_free(esc);
00399     objunlock(curl_isinit);
00400     objunref(curl_isinit);
00401     return ret;
00402 }
```

12.25.3.13 char* url_unescape (char * *url*)

UN escape and return the url.

Parameters

url	URL to un escape

Returns

A malloc()'d URL that needs to be free()'d

Definition at line 407 of file curl.c.

References curlinit(), objlock(), objunlock(), and objunref().

```
00407                                       {
00408     char *uesc;
00409     char *ret = NULL;
00410 
00411     if (!curlinit()) {
00412         return NULL;
00413     }
00414 
00415     objlock(curl_isinit);
00416     uesc = curl_easy_unescape(curl, url, 0, 0);
00417     if (uesc) {
00418         ret = strdup(uesc);
00419     }
00420     curl_free(uesc);
00421     objunlock(curl_isinit);
00422     objunref(curl_isinit);
00423     return ret;
00424 }
```

12.26 Zlib Interface

Simplified implementation of zlib functions.

Files

- file zlib.c

  Simplified implementation of zlib functions.

Data Structures

- struct zobj

  Zlib buffer used for compression and decompression.

Functions

- struct zobj ∗ zcompress (uint8_t ∗buff, uint16_t len, uint8_t level)

  Allocate a buffer and return it with compressed data.
- void zuncompress (struct zobj ∗buff, uint8_t ∗obuff)

  Uncompress zobj buffer to buffer.
- int is_gzip (uint8_t ∗buf, int buf_size)

  check a buffer if it contains gzip magic
- uint8_t ∗ gzinflatebuf (uint8_t ∗buf_in, int buf_size, uint32_t ∗len)

  Ungzip a buffer.

12.26.1 Detailed Description

Simplified implementation of zlib functions.

12.26.2 Function Documentation

12.26.2.1 uint8_t∗ gzinflatebuf (uint8_t ∗ *buf_in,* int *buf_size,* uint32_t ∗ *len*)

Ungzip a buffer.

Parameters

buf_in	Buffer to inflate.
buf_size	Size of buf_in buffer.
len	Pointer that will contain the uncompressed data length.

Returns

Uncompressed data in a buffer or NULL on error.

Definition at line 101 of file zlib.c.

Referenced by curl_ungzip().

```
00101                                                                   {
00102     z_stream zdat;
00103     uint8_t *buf = NULL, *tmp;
00104     int res;
00105
00106     zdat.opaque = NULL;
```

```
00107     zdat.zalloc = NULL;
00108     zdat.zfree = NULL;
00109
00110     zdat.next_in = buf_in;
00111     zdat.avail_in = buf_size;
00112     zdat.next_out = buf;
00113     zdat.avail_out = 0;
00114     zdat.total_out = 0;
00115
00116     if (inflateInit2(&zdat, 31)) {
00117         return NULL;
00118     }
00119
00120     do {
00121         if (!(tmp = realloc(buf,zdat.total_out + (zdat.avail_in * 5) + 1))) {
00122             res = Z_MEM_ERROR;
00123             break;
00124         } else {
00125             buf = tmp;
00126         }
00127         buf[zdat.total_out] = '\0';
00128         zdat.next_out = &buf[zdat.total_out];
00129         zdat.avail_out += zdat.avail_in * 5;
00130     } while ((res = inflate(&zdat, Z_NO_FLUSH)) == Z_OK);
00131
00132     if (res == Z_STREAM_END) {
00133         buf = realloc(buf, zdat.total_out);
00134         *len = zdat.total_out;
00135     } else {
00136         free(buf);
00137         *len = 0;
00138         buf = NULL;
00139     }
00140     inflateEnd(&zdat);
00141
00142     return buf;
00143 }
```

12.26.2.2 int is_gzip (uint8_t * *buf*, int *buf_size*)

check a buffer if it contains gzip magic

Parameters

buf	buffer to check.
buf_size	buffer len it must be more than 4.

Returns

non zero value if the buffer contains gzip data

Definition at line 85 of file zlib.c.

Referenced by curl_ungzip().

```
00085                                           {
00086     if (buf_size < 4) {
00087         return 0;
00088     }
00089     if (memcmp(buf, gzipMagicBytes, 4)) {
00090         return 0;
00091     }
00092     return 1;
00093 }
```

12.26.2.3 struct zobj* zcompress (uint8_t * *buff*, uint16_t *len*, uint8_t *level*)

Allocate a buffer and return it with compressed data.

Parameters

buff	Buffer to compress.
len	Length of the buffer.
level	Compression level.

Returns

reference to zobj data structure containing compressed data or NULL on error.

Definition at line 47 of file zlib.c.

References zobj::buff, objalloc(), zobj::olen, and zobj::zlen.

```
00047                                                                   {
00048     struct zobj *ret;
00049
00050     if (!(ret = objalloc(sizeof(*ret), zobj_free))) {
00051         return (NULL);
00052     }
00053
00054     ret->zlen = compressBound(len);
00055     ret->olen = len;
00056
00057     if (!(ret->buff = malloc(ret->zlen))) {
00058         return (NULL);
00059     }
00060     compress2(ret->buff, (uLongf *)&ret->zlen, buff, len, level);
00061
00062     return (ret);
00063 }
```

12.26.2.4 void zuncompress (struct zobj * *buff*, uint8_t * *obuff*)

Uncompress zobj buffer to buffer.

Parameters

buff	Compressed buffer to uncompress.
obuff	Buffer to uncompress too.

Warning

obuff needs to be large enough to contain the data.

Todo Implement this without needing original buff len using inflate

Definition at line 71 of file zlib.c.

References zobj::buff, zobj::olen, and zobj::zlen.

```
00071                                                   {
00072     uLongf olen = buff->olen;
00073
00074     if (!obuff) {
00075         return;
00076     }
00077
00078     uncompress(obuff, &olen, buff->buff, buff->zlen);
00079 }
```

12.27 Burtle Bob hash algorythim.

lookup3.c, by Bob Jenkins, May 2006, Public Domain (Original Documentation)

Files

- file lookup3.c

  by Bob Jenkins, May 2006, Public Domain.

Macros

- #define JHASH_INITVAL 0xdeadbeef

  Default init value for hash function
  easter egg copied from <linux/jhash.h>
- #define jenhash(key, length, initval) hashlittle(key, length, (initval) ? initval : JHASH_INITVAL);

  Define jenhash as hashlittle on big endian it should be hashbig.
- #define HASH_LITTLE_ENDIAN 0
- #define HASH_BIG_ENDIAN 0
- #define hashsize(n) ((uint32_t)1<<(n))
- #define hashmask(n) (hashsize(n)-1)
- #define rot(x, k) (((x)<<(k)) | ((x)>>(32-(k))))
- #define mix(a, b, c)

  mix 3 32-bit values reversibly
- #define final(a, b, c)

  final mixing of 3 32-bit values (a,b,c) into c

Functions

- uint32_t hashword (const uint32_t *k, size_t length, uint32_t initval)

  hash a variable-length key into a 32-bit value (Big Endian)
- void hashword2 (const uint32_t *k, size_t length, uint32_t *pc, uint32_t *pb)

  same as hashword(), but take two seeds and return two 32-bit values
- uint32_t hashlittle (const void *key, size_t length, uint32_t initval)

  hash a variable-length key into a 32-bit value (Little Endian)
- void hashlittle2 (const void *key, size_t length, uint32_t *pc, uint32_t *pb)

  return 2 32-bit hash values.
- uint32_t hashbig (const void *key, size_t length, uint32_t initval)

  This is the same as hashword() on big-endian machines.

12.27.1 Detailed Description

lookup3.c, by Bob Jenkins, May 2006, Public Domain (Original Documentation)

```
--------------------------------------------------------------------------------
lookup3.c, by Bob Jenkins, May 2006, Public Domain.

These are functions for producing 32-bit hashes for hash table lookup.
hashword(), hashlittle(), hashlittle2(), hashbig(), mix(), and final()
are externally useful functions.  Routines to test the hash are included
if SELF_TEST is defined.  You can use this free for any purpose.  It's in
the public domain.  It has no warranty.

You probably want to use hashlittle().  hashlittle() and hashbig()
hash byte arrays.  hashlittle() is is faster than hashbig() on
```

```
little-endian machines.  Intel and AMD are little-endian machines.
On second thought, you probably want hashlittle2(), which is identical to
hashlittle() except it returns two 32-bit hashes for the price of one.
You could implement hashbig2() if you wanted but I haven't bothered here.

If you want to find a hash of, say, exactly 7 integers, do
  a = i1;  b = i2;  c = i3;
  mix(a,b,c);
  a += i4; b += i5; c += i6;
  mix(a,b,c);
  a += i7;
  final(a,b,c);
then use c as the hash value.  If you have a variable length array of
4-byte integers to hash, use hashword().  If you have a byte array (like
a character string), use hashlittle().  If you have several byte arrays, or
a mix of things, see the comments above hashlittle().

Why is this so big?  I read 12 bytes at a time into 3 4-byte integers,
then mix those integers.  This is fast (you can do a lot more thorough
mixing with 12*3 instructions on 3 integers than you can with 3 instructions
on 1 byte), but shoehorning those bytes into integers efficiently is messy.
-------------------------------------------------------------------------------
```

12.27.2 Macro Definition Documentation

12.27.2.1 #define final(*a, b, c*)

Value:

```
{ \
  c ^= b; c -= rot(b,14); \
  a ^= c; a -= rot(c,11); \
  b ^= a; b -= rot(a,25); \
  c ^= b; c -= rot(b,16); \
  a ^= c; a -= rot(c,4);  \
  b ^= a; b -= rot(a,14); \
  c ^= b; c -= rot(b,24); \
}
```

final mixing of 3 32-bit values (a,b,c) into c

```
-------------------------------------------------------------------------------
final -- final mixing of 3 32-bit values (a,b,c) into c

Pairs of (a,b,c) values differing in only a few bits will usually
produce values of c that look totally different.  This was tested for
* pairs that differed by one bit, by two bits, in any combination
  of top bits of (a,b,c), or in any combination of bottom bits of
  (a,b,c).
* "differ" is defined as +, -, ^, or ~^.  For + and -, I transformed
  the output delta to a Gray code (a^(a>>1)) so a string of 1's (as
  is commonly produced by subtraction) look like a single 1-bit
  difference.
* the base values were pseudorandom, all zero but one bit set, or
  all zero plus a counter that starts at zero.

These constants passed:
 14 11 25 16 4 14 24
 12 14 25 16 4 14 24
and these came close:
  4  8 15 26 3 22 24
 10  8 15 26 3 22 24
 11  8 15 26 3 22 24
-------------------------------------------------------------------------------
```

Definition at line 158 of file lookup3.c.

12.27.2.2 #define HASH_BIG_ENDIAN 0

Definition at line 70 of file lookup3.c.

Referenced by hashbig().

12.27.2.3 #define HASH_LITTLE_ENDIAN 0

Definition at line 69 of file lookup3.c.

Referenced by hashlittle(), and hashlittle2().

12.27.2.4 #define hashmask(*n*) (hashsize(n)-1)

Definition at line 74 of file lookup3.c.

12.27.2.5 #define hashsize(*n*) ((uint32_t)1<<(n))

Definition at line 73 of file lookup3.c.

12.27.2.6 #define jenhash(*key, length, initval*) hashlittle(key, length, (initval) ? initval : JHASH_INITVAL);

Define jenhash as hashlittle on big endian it should be hashbig.

Definition at line 914 of file dtsapp.h.

12.27.2.7 #define JHASH_INITVAL 0xdeadbeef

Default init value for hash function

easter egg copied from <linux/jhash.h>

Definition at line 909 of file dtsapp.h.

12.27.2.8 #define mix(*a, b, c*)

Value:

```
{ \
  a -= c;  a ^= rot(c, 4);  c += b; \
  b -= a;  b ^= rot(a, 6);  a += c; \
  c -= b;  c ^= rot(b, 8);  b += a; \
  a -= c;  a ^= rot(c,16);  c += b; \
  b -= a;  b ^= rot(a,19);  a += c; \
  c -= b;  c ^= rot(b, 4);  b += a; \
}
```

mix 3 32-bit values reversibly

```
-------------------------------------------------------------------------------
mix -- mix 3 32-bit values reversibly.

This is reversible, so any information in (a,b,c) before mix() is
still in (a,b,c) after mix().

If four pairs of (a,b,c) inputs are run through mix(), or through
mix() in reverse, there are at least 32 bits of the output that
are sometimes the same for one pair and different for another pair.
This was tested for:
* pairs that differed by one bit, by two bits, in any combination
```

```
  of top bits of (a,b,c), or in any combination of bottom bits of
  (a,b,c).
* "differ" is defined as +, -, ^, or ~^.  For + and -, I transformed
  the output delta to a Gray code (a^(a>>1)) so a string of 1's (as
  is commonly produced by subtraction) look like a single 1-bit
  difference.
* the base values were pseudorandom, all zero but one bit set, or
  all zero plus a counter that starts at zero.

Some k values for my "a-=c; a^=rot(c,k); c+=b;" arrangement that
satisfy this are
    4  6  8 16 19  4
    9 15  3 18 27 15
   14  9  3  7 17  3
Well, "9 15 3 18 27 15" didn't quite get 32 bits diffing
for "differ" defined as + with a one-bit base and a two-bit delta.  I
used http://burtleburtle.net/bob/hash/avalanche.html to choose
the operations, constants, and arrangements of the variables.

This does not achieve avalanche.  There are input bits of (a,b,c)
that fail to affect some output bits of (a,b,c), especially of a.  The
most thoroughly mixed value is c, but it doesn't really even achieve
avalanche in c.

This allows some parallelism.  Read-after-writes are good at doubling
the number of bits affected, so the goal of mixing pulls in the opposite
direction as the goal of parallelism.  I did what I could.  Rotates
seem to cost as much as shifts on every machine I could lay my hands
on, and rotates are much kinder to the top and bottom bits, so I used
rotates.
-------------------------------------------------------------------------------
```

Definition at line 122 of file lookup3.c.

Referenced by hashbig(), hashlittle(), hashlittle2(), hashword(), and hashword2().

12.27.2.9 #define rot(*x*, *k*) (((x)<<(k)) | ((x)>>(32-(k))))

Definition at line 75 of file lookup3.c.

12.27.3 Function Documentation

12.27.3.1 uint32_t hashbig (const void * *key*, size_t *length*, uint32_t *initval*)

This is the same as hashword() on big-endian machines.

```
 * hashbig():
 * This is the same as hashword() on big-endian machines.  It is different
 * from hashlittle() on all machines.  hashbig() takes advantage of
 * big-endian byte ordering.
```

Definition at line 862 of file lookup3.c.

References HASH_BIG_ENDIAN, and mix.

```
00862                                                                   {
00863     uint32_t a,b,c;
00864     union {
00865         const void *ptr;
00866         size_t i;
00867     } u; /* to cast key to (size_t) happily */
00868
00869     /* Set up the internal state */
00870     a = b = c = 0xdeadbeef + ((uint32_t)length) + initval;
00871
00872     u.ptr = key;
00873     if (HASH_BIG_ENDIAN && ((u.i & 0x3) == 0)) {
00874         const uint32_t *k = (const uint32_t *)key;         /* read 32-bit chunks */
```

```
00875 #ifdef VALGRIND
00876         const uint8_t  *k8;
00877 #endif
00878         /*------ all but last block: aligned reads and affect 32 bits of (a,b,c) */
00879         while (length > 12) {
00880             a += k[0];
00881             b += k[1];
00882             c += k[2];
00883             mix(a,b,c);
00884             length -= 12;
00885             k += 3;
00886         }
00887 
00888         /*------------------------------ handle the last (probably partial) block */
00889         /*
00890          * "k[2]<<8" actually reads beyond the end of the string, but
00891          * then shifts out the part it's not allowed to read.  Because the
00892          * string is aligned, the illegal read is in the same word as the
00893          * rest of the string.  Every machine with memory protection I've seen
00894          * does it on word boundaries, so is OK with this.  But VALGRIND will
00895          * still catch it and complain.  The masking trick does make the hash
00896          * noticably faster for short strings (like English words).
00897          */
00898 #ifndef VALGRIND
00899 
00900         switch(length) {
00901             case 12:
00902                 c+=k[2];
00903                 b+=k[1];
00904                 a+=k[0];
00905                 break;
00906             case 11:
00907                 c+=k[2]&0xffffff00;
00908                 b+=k[1];
00909                 a+=k[0];
00910                 break;
00911             case 10:
00912                 c+=k[2]&0xffff0000;
00913                 b+=k[1];
00914                 a+=k[0];
00915                 break;
00916             case 9 :
00917                 c+=k[2]&0xff000000;
00918                 b+=k[1];
00919                 a+=k[0];
00920                 break;
00921             case 8 :
00922                 b+=k[1];
00923                 a+=k[0];
00924                 break;
00925             case 7 :
00926                 b+=k[1]&0xffffff00;
00927                 a+=k[0];
00928                 break;
00929             case 6 :
00930                 b+=k[1]&0xffff0000;
00931                 a+=k[0];
00932                 break;
00933             case 5 :
00934                 b+=k[1]&0xff000000;
00935                 a+=k[0];
00936                 break;
00937             case 4 :
00938                 a+=k[0];
00939                 break;
00940             case 3 :
00941                 a+=k[0]&0xffffff00;
00942                 break;
00943             case 2 :
00944                 a+=k[0]&0xffff0000;
00945                 break;
00946             case 1 :
00947                 a+=k[0]&0xff000000;
00948                 break;
00949             case 0 :
00950                 return (c);              /* zero length strings require no mixing */
00951         }
00952 
00953 #else  /* make valgrind happy */
00954 
00955         k8 = (const uint8_t *)k;
00956         switch(length) {                 /* all the case statements fall through */
00957             case 12:
00958                 c+=k[2];
00959                 b+=k[1];
00960                 a+=k[0];
00961                 break;
```

```
00962             case 11:
00963                 c+=((uint32_t)k8[10])<<8;  /* fall through */
00964             case 10:
00965                 c+=((uint32_t)k8[9])<<16;  /* fall through */
00966             case 9 :
00967                 c+=((uint32_t)k8[8])<<24;  /* fall through */
00968             case 8 :
00969                 b+=k[1];
00970                 a+=k[0];
00971                 break;
00972             case 7 :
00973                 b+=((uint32_t)k8[6])<<8;   /* fall through */
00974             case 6 :
00975                 b+=((uint32_t)k8[5])<<16;  /* fall through */
00976             case 5 :
00977                 b+=((uint32_t)k8[4])<<24;  /* fall through */
00978             case 4 :
00979                 a+=k[0];
00980                 break;
00981             case 3 :
00982                 a+=((uint32_t)k8[2])<<8;   /* fall through */
00983             case 2 :
00984                 a+=((uint32_t)k8[1])<<16;  /* fall through */
00985             case 1 :
00986                 a+=((uint32_t)k8[0])<<24;
00987                 break;
00988             case 0 :
00989                 return c;
00990         }
00991
00992 #endif /* !VALGRIND */
00993
00994     } else {                        /* need to read the key one byte at a time */
00995         const uint8_t *k = (const uint8_t *)key;
00996
00997         /*--------------- all but the last block: affect some 32 bits of (a,b,c) */
00998         while (length > 12) {
00999             a += ((uint32_t)k[0])<<24;
01000             a += ((uint32_t)k[1])<<16;
01001             a += ((uint32_t)k[2])<<8;
01002             a += ((uint32_t)k[3]);
01003             b += ((uint32_t)k[4])<<24;
01004             b += ((uint32_t)k[5])<<16;
01005             b += ((uint32_t)k[6])<<8;
01006             b += ((uint32_t)k[7]);
01007             c += ((uint32_t)k[8])<<24;
01008             c += ((uint32_t)k[9])<<16;
01009             c += ((uint32_t)k[10])<<8;
01010             c += ((uint32_t)k[11]);
01011             mix(a,b,c);
01012             length -= 12;
01013             k += 12;
01014         }
01015
01016         /*-------------------------------- last block: affect all 32 bits of (c) */
01017         switch(length) {                   /* all the case statements fall through */
01018             case 12:
01019                 c+=k[11];
01020                 /* no break */
01021             case 11:
01022                 c+=((uint32_t)k[10])<<8;
01023                 /* no break */
01024             case 10:
01025                 c+=((uint32_t)k[9])<<16;
01026                 /* no break */
01027             case 9 :
01028                 c+=((uint32_t)k[8])<<24;
01029                 /* no break */
01030             case 8 :
01031                 b+=k[7];
01032                 /* no break */
01033             case 7 :
01034                 b+=((uint32_t)k[6])<<8;
01035                 /* no break */
01036             case 6 :
01037                 b+=((uint32_t)k[5])<<16;
01038                 /* no break */
01039             case 5 :
01040                 b+=((uint32_t)k[4])<<24;
01041                 /* no break */
01042             case 4 :
01043                 a+=k[3];
01044                 /* no break */
01045             case 3 :
01046                 a+=((uint32_t)k[2])<<8;
01047                 /* no break */
01048             case 2 :
```

```
01049                 a+=((uint32_t)k[1])<<16;
01050                 /* no break */
01051             case 1 :
01052                 a+=((uint32_t)k[0])<<24;
01053                 break;
01054             case 0 :
01055                 return (c);
01056         }
01057     }
01058
01059     final(a,b,c);
01060     return (c);
01061 }
```

12.27.3.2 uint32_t hashlittle (const void * *key*, size_t *length*, uint32_t *initval*)

hash a variable-length key into a 32-bit value (Little Endian)

```
--------------------------------------------------------------------------------
hashlittle() -- hash a variable-length key into a 32-bit value
  k        : the key (the unaligned variable-length array of bytes)
  length   : the length of the key, counting by bytes
  initval  : can be any 4-byte value
Returns a 32-bit value.  Every bit of the key affects every bit of
the return value.  Two keys differing by one or two bits will have
totally different hash values.

The best hash table sizes are powers of 2.  There is no need to do
mod a prime (mod is sooo slow!).  If you need less than 32 bits,
use a bitmask.  For example, if you need only 10 bits, do
  h = (h & hashmask(10));
In which case, the hash table should have hashsize(10) elements.

If you are hashing n strings (uint8_t **)k, do it like this:
  for (i=0, h=0; i<n; ++i) h = hashlittle( k[i], len[i], h);

By Bob Jenkins, 2006.  bob_jenkins@burtleburtle.net.  You may use this
code any way you wish, private, educational, or commercial.  It's free.

Use for hash table lookup, or anything where one collision in 2^^32 is
acceptable.  Do NOT use for cryptographic purposes.
--------------------------------------------------------------------------------
```

Definition at line 298 of file lookup3.c.

References HASH_LITTLE_ENDIAN, and mix.

```
00298                                                                   {
00299     uint32_t a,b,c;                                          /* internal state */
00300     union {
00301         const void *ptr;
00302         size_t i;
00303     } u;     /* needed for Mac Powerbook G4 */
00304
00305     /* Set up the internal state */
00306     a = b = c = 0xdeadbeef + ((uint32_t)length) + initval;
00307
00308     u.ptr = key;
00309     if (HASH_LITTLE_ENDIAN && ((u.i & 0x3) == 0)) {
00310         const uint32_t *k = (const uint32_t *)key;         /* read 32-bit chunks */
00311 #ifdef VALGRIND
00312         const uint8_t  *k8;
00313 #endif
00314         /*------ all but last block: aligned reads and affect 32 bits of (a,b,c) */
00315         while (length > 12) {
00316             a += k[0];
00317             b += k[1];
00318             c += k[2];
00319             mix(a,b,c);
00320             length -= 12;
00321             k += 3;
00322         }
00323
00324         /*---------------------------- handle the last (probably partial) block */
00325         /*
```

```
00326          * "k[2]&0xffffff" actually reads beyond the end of the string, but
00327          * then masks off the part it's not allowed to read.  Because the
00328          * string is aligned, the masked-off tail is in the same word as the
00329          * rest of the string.  Every machine with memory protection I've seen
00330          * does it on word boundaries, so is OK with this.  But VALGRIND will
00331          * still catch it and complain.  The masking trick does make the hash
00332          * noticably faster for short strings (like English words).
00333          */
00334 #ifndef VALGRIND
00335
00336         switch(length) {
00337             case 12:
00338                 c+=k[2];
00339                 b+=k[1];
00340                 a+=k[0];
00341                 break;
00342             case 11:
00343                 c+=k[2]&0xffffff;
00344                 b+=k[1];
00345                 a+=k[0];
00346                 break;
00347             case 10:
00348                 c+=k[2]&0xffff;
00349                 b+=k[1];
00350                 a+=k[0];
00351                 break;
00352             case 9 :
00353                 c+=k[2]&0xff;
00354                 b+=k[1];
00355                 a+=k[0];
00356                 break;
00357             case 8 :
00358                 b+=k[1];
00359                 a+=k[0];
00360                 break;
00361             case 7 :
00362                 b+=k[1]&0xffffff;
00363                 a+=k[0];
00364                 break;
00365             case 6 :
00366                 b+=k[1]&0xffff;
00367                 a+=k[0];
00368                 break;
00369             case 5 :
00370                 b+=k[1]&0xff;
00371                 a+=k[0];
00372                 break;
00373             case 4 :
00374                 a+=k[0];
00375                 break;
00376             case 3 :
00377                 a+=k[0]&0xffffff;
00378                 break;
00379             case 2 :
00380                 a+=k[0]&0xffff;
00381                 break;
00382             case 1 :
00383                 a+=k[0]&0xff;
00384                 break;
00385             case 0 :
00386                 return (c);              /* zero length strings require no mixing */
00387         }
00388
00389 #else /* make valgrind happy */
00390
00391         k8 = (const uint8_t *)k;
00392         switch(length) {
00393             case 12:
00394                 c+=k[2];
00395                 b+=k[1];
00396                 a+=k[0];
00397                 break;
00398             case 11:
00399                 c+=((uint32_t)k8[10])<<16;  /* fall through */
00400             case 10:
00401                 c+=((uint32_t)k8[9])<<8;    /* fall through */
00402             case 9 :
00403                 c+=k8[8];                   /* fall through */
00404             case 8 :
00405                 b+=k[1];
00406                 a+=k[0];
00407                 break;
00408             case 7 :
00409                 b+=((uint32_t)k8[6])<<16;   /* fall through */
00410             case 6 :
00411                 b+=((uint32_t)k8[5])<<8;    /* fall through */
00412             case 5 :
```

```
00413                 b+=k8[4];                   /* fall through */
00414             case 4 :
00415                 a+=k[0];
00416                 break;
00417             case 3 :
00418                 a+=((uint32_t)k8[2])<<16;   /* fall through */
00419             case 2 :
00420                 a+=((uint32_t)k8[1])<<8;    /* fall through */
00421             case 1 :
00422                 a+=k8[0];
00423                 break;
00424             case 0 :
00425                 return c;
00426         }
00427
00428 #endif /* !valgrind */
00429
00430     } else
00431         if (HASH_LITTLE_ENDIAN && ((u.i & 0x1) == 0)) {
00432             const uint16_t *k = (const uint16_t *)key;         /* read 16-bit chunks */
00433             const uint8_t  *k8;
00434
00435             /*--------------- all but last block: aligned reads and different mixing */
00436             while (length > 12) {
00437                 a += k[0] + (((uint32_t)k[1])<<16);
00438                 b += k[2] + (((uint32_t)k[3])<<16);
00439                 c += k[4] + (((uint32_t)k[5])<<16);
00440                 mix(a,b,c);
00441                 length -= 12;
00442                 k += 6;
00443             }
00444
00445             /*----------------------------- handle the last (probably partial) block */
00446             k8 = (const uint8_t *)k;
00447             switch(length) {
00448                 case 12:
00449                     c+=k[4]+(((uint32_t)k[5])<<16);
00450                     b+=k[2]+(((uint32_t)k[3])<<16);
00451                     a+=k[0]+(((uint32_t)k[1])<<16);
00452                     break;
00453                 case 11:
00454                     c+=((uint32_t)k8[10])<<16;     /* fall through */
00455                     /* no break */
00456                 case 10:
00457                     c+=k[4];
00458                     b+=k[2]+(((uint32_t)k[3])<<16);
00459                     a+=k[0]+(((uint32_t)k[1])<<16);
00460                     break;
00461                 case 9 :
00462                     c+=k8[8];                      /* fall through */
00463                     /* no break */
00464                 case 8 :
00465                     b+=k[2]+(((uint32_t)k[3])<<16);
00466                     a+=k[0]+(((uint32_t)k[1])<<16);
00467                     break;
00468                 case 7 :
00469                     b+=((uint32_t)k8[6])<<16;      /* fall through */
00470                     /* no break */
00471                 case 6 :
00472                     b+=k[2];
00473                     a+=k[0]+(((uint32_t)k[1])<<16);
00474                     break;
00475                 case 5 :
00476                     b+=k8[4];                      /* fall through */
00477                     /* no break */
00478                 case 4 :
00479                     a+=k[0]+(((uint32_t)k[1])<<16);
00480                     break;
00481                 case 3 :
00482                     a+=((uint32_t)k8[2])<<16;      /* fall through */
00483                     /* no break */
00484                 case 2 :
00485                     a+=k[0];
00486                     break;
00487                 case 1 :
00488                     a+=k8[0];
00489                     break;
00490                 case 0 :
00491                     return (c);                     /* zero length requires no mixing */
00492             }
00493
00494         } else {                        /* need to read the key one byte at a time */
00495             const uint8_t *k = (const uint8_t *)key;
00496
00497             /*--------------- all but the last block: affect some 32 bits of (a,b,c) */
00498             while (length > 12) {
00499                 a += k[0];
```

```
00500                 a += ((uint32_t)k[1])<<8;
00501                 a += ((uint32_t)k[2])<<16;
00502                 a += ((uint32_t)k[3])<<24;
00503                 b += k[4];
00504                 b += ((uint32_t)k[5])<<8;
00505                 b += ((uint32_t)k[6])<<16;
00506                 b += ((uint32_t)k[7])<<24;
00507                 c += k[8];
00508                 c += ((uint32_t)k[9])<<8;
00509                 c += ((uint32_t)k[10])<<16;
00510                 c += ((uint32_t)k[11])<<24;
00511                 mix(a,b,c);
00512                 length -= 12;
00513                 k += 12;
00514             }
00515 
00516             /*-------------------------------- last block: affect all 32 bits of (c) */
00517             switch(length) {                   /* all the case statements fall through */
00518                 case 12:
00519                     c+=((uint32_t)k[11])<<24;
00520                     /* no break */
00521                 case 11:
00522                     c+=((uint32_t)k[10])<<16;
00523                     /* no break */
00524                 case 10:
00525                     c+=((uint32_t)k[9])<<8;
00526                     /* no break */
00527                 case 9 :
00528                     c+=k[8];
00529                     /* no break */
00530                 case 8 :
00531                     b+=((uint32_t)k[7])<<24;
00532                     /* no break */
00533                 case 7 :
00534                     b+=((uint32_t)k[6])<<16;
00535                     /* no break */
00536                 case 6 :
00537                     b+=((uint32_t)k[5])<<8;
00538                     /* no break */
00539                 case 5 :
00540                     b+=k[4];
00541                     /* no break */
00542                 case 4 :
00543                     a+=((uint32_t)k[3])<<24;
00544                     /* no break */
00545                 case 3 :
00546                     a+=((uint32_t)k[2])<<16;
00547                     /* no break */
00548                 case 2 :
00549                     a+=((uint32_t)k[1])<<8;
00550                     /* no break */
00551                 case 1 :
00552                     a+=k[0];
00553                     break;
00554                 case 0 :
00555                     return (c);
00556             }
00557         }
00558 
00559     final(a,b,c);
00560     return (c);
00561 }
```

12.27.3.3 void hashlittle2 (const void * *key*, size_t *length*, uint32_t * *pc*, uint32_t * *pb*)

return 2 32-bit hash values.

```
* hashlittle2: return 2 32-bit hash values
*
* This is identical to hashlittle(), except it returns two 32-bit hash
* values instead of just one.  This is good enough for hash table
* lookup with 2^^64 buckets, or if you want a second hash if you're not
* happy with the first, or if you want a probably-unique 64-bit ID for
* the key.  *pc is better mixed than *pb, so use *pc first.  If you want
* a 64-bit value do something like "*pc + (((uint64_t)*pb)<<32)".
```

Definition at line 574 of file lookup3.c.

References HASH_LITTLE_ENDIAN, and mix.

```
00578                     {      /* IN: secondary initval, OUT: secondary hash */
00579     uint32_t a,b,c;                                          /* internal state */
00580     union {
00581         const void *ptr;
00582         size_t i;
00583     } u;     /* needed for Mac Powerbook G4 */
00584
00585     /* Set up the internal state */
00586     a = b = c = 0xdeadbeef + ((uint32_t)length) + *pc;
00587     c += *pb;
00588
00589     u.ptr = key;
00590     if (HASH_LITTLE_ENDIAN && ((u.i & 0x3) == 0)) {
00591         const uint32_t *k = (const uint32_t *)key;         /* read 32-bit chunks */
00592 #ifdef VALGRIND
00593         const uint8_t  *k8;
00594 #endif
00595
00596         /*------ all but last block: aligned reads and affect 32 bits of (a,b,c) */
00597         while (length > 12) {
00598             a += k[0];
00599             b += k[1];
00600             c += k[2];
00601             mix(a,b,c);
00602             length -= 12;
00603             k += 3;
00604         }
00605
00606         /*----------------------------- handle the last (probably partial) block */
00607         /*
00608          * "k[2]&0xffffff" actually reads beyond the end of the string, but
00609          * then masks off the part it's not allowed to read.  Because the
00610          * string is aligned, the masked-off tail is in the same word as the
00611          * rest of the string.  Every machine with memory protection I've seen
00612          * does it on word boundaries, so is OK with this.  But VALGRIND will
00613          * still catch it and complain.  The masking trick does make the hash
00614          * noticably faster for short strings (like English words).
00615          */
00616 #ifndef VALGRIND
00617
00618         switch(length) {
00619             case 12:
00620                 c+=k[2];
00621                 b+=k[1];
00622                 a+=k[0];
00623                 break;
00624             case 11:
00625                 c+=k[2]&0xffffff;
00626                 b+=k[1];
00627                 a+=k[0];
00628                 break;
00629             case 10:
00630                 c+=k[2]&0xffff;
00631                 b+=k[1];
00632                 a+=k[0];
00633                 break;
00634             case 9 :
00635                 c+=k[2]&0xff;
00636                 b+=k[1];
00637                 a+=k[0];
00638                 break;
00639             case 8 :
00640                 b+=k[1];
00641                 a+=k[0];
00642                 break;
00643             case 7 :
00644                 b+=k[1]&0xffffff;
00645                 a+=k[0];
00646                 break;
00647             case 6 :
00648                 b+=k[1]&0xffff;
00649                 a+=k[0];
00650                 break;
00651             case 5 :
00652                 b+=k[1]&0xff;
00653                 a+=k[0];
00654                 break;
00655             case 4 :
00656                 a+=k[0];
00657                 break;
00658             case 3 :
00659                 a+=k[0]&0xffffff;
00660                 break;
00661             case 2 :
00662                 a+=k[0]&0xffff;
00663                 break;
00664             case 1 :
```

```
00665                 a+=k[0]&0xff;
00666                 break;
00667             case 0 :
00668                 *pc=c;
00669                 *pb=b;
00670                 return;  /* zero length strings require no mixing */
00671         }
00672
00673 #else /* make valgrind happy */
00674
00675         k8 = (const uint8_t *)k;
00676         switch(length) {
00677             case 12:
00678                 c+=k[2];
00679                 b+=k[1];
00680                 a+=k[0];
00681                 break;
00682             case 11:
00683                 c+=((uint32_t)k8[10])<<16;  /* fall through */
00684             case 10:
00685                 c+=((uint32_t)k8[9])<<8;    /* fall through */
00686             case 9 :
00687                 c+=k8[8];                   /* fall through */
00688             case 8 :
00689                 b+=k[1];
00690                 a+=k[0];
00691                 break;
00692             case 7 :
00693                 b+=((uint32_t)k8[6])<<16;   /* fall through */
00694             case 6 :
00695                 b+=((uint32_t)k8[5])<<8;    /* fall through */
00696             case 5 :
00697                 b+=k8[4];                   /* fall through */
00698             case 4 :
00699                 a+=k[0];
00700                 break;
00701             case 3 :
00702                 a+=((uint32_t)k8[2])<<16;   /* fall through */
00703             case 2 :
00704                 a+=((uint32_t)k8[1])<<8;    /* fall through */
00705             case 1 :
00706                 a+=k8[0];
00707                 break;
00708             case 0 :
00709                 *pc=c;
00710                 *pb=b;
00711                 return;  /* zero length strings require no mixing */
00712         }
00713
00714 #endif /* !valgrind */
00715
00716     } else
00717         if (HASH_LITTLE_ENDIAN && ((u.i & 0x1) == 0)) {
00718             const uint16_t *k = (const uint16_t *)key;         /* read 16-bit chunks */
00719             const uint8_t  *k8;
00720
00721             /*--------------- all but last block: aligned reads and different mixing */
00722             while (length > 12) {
00723                 a += k[0] + (((uint32_t)k[1])<<16);
00724                 b += k[2] + (((uint32_t)k[3])<<16);
00725                 c += k[4] + (((uint32_t)k[5])<<16);
00726                 mix(a,b,c);
00727                 length -= 12;
00728                 k += 6;
00729             }
00730
00731             /*----------------------------- handle the last (probably partial) block */
00732             k8 = (const uint8_t *)k;
00733             switch(length) {
00734                 case 12:
00735                     c+=k[4]+(((uint32_t)k[5])<<16);
00736                     b+=k[2]+(((uint32_t)k[3])<<16);
00737                     a+=k[0]+(((uint32_t)k[1])<<16);
00738                     break;
00739                 case 11:
00740                     c+=((uint32_t)k8[10])<<16;     /* fall through */
00741                     /* no break */
00742                 case 10:
00743                     c+=k[4];
00744                     b+=k[2]+(((uint32_t)k[3])<<16);
00745                     a+=k[0]+(((uint32_t)k[1])<<16);
00746                     break;
00747                 case 9 :
00748                     c+=k8[8];                      /* fall through */
00749                     /* no break */
00750                 case 8 :
00751                     b+=k[2]+(((uint32_t)k[3])<<16);
```

```
00752                     a+=k[0]+(((uint32_t)k[1])<<16);
00753                     break;
00754                 case 7 :
00755                     b+=((uint32_t)k8[6])<<16;      /* fall through */
00756                     /* no break */
00757                 case 6 :
00758                     b+=k[2];
00759                     a+=k[0]+(((uint32_t)k[1])<<16);
00760                     break;
00761                 case 5 :
00762                     b+=k8[4];                      /* fall through */
00763                     /* no break */
00764                 case 4 :
00765                     a+=k[0]+(((uint32_t)k[1])<<16);
00766                     break;
00767                 case 3 :
00768                     a+=((uint32_t)k8[2])<<16;      /* fall through */
00769                     /* no break */
00770                 case 2 :
00771                     a+=k[0];
00772                     break;
00773                 case 1 :
00774                     a+=k8[0];
00775                     break;
00776                 case 0 :
00777                     *pc=c;
00778                     *pb=b;
00779                     return;  /* zero length strings require no mixing */
00780             }
00781
00782         } else {                        /* need to read the key one byte at a time */
00783             const uint8_t *k = (const uint8_t *)key;
00784
00785             /*--------------- all but the last block: affect some 32 bits of (a,b,c) */
00786             while (length > 12) {
00787                 a += k[0];
00788                 a += ((uint32_t)k[1])<<8;
00789                 a += ((uint32_t)k[2])<<16;
00790                 a += ((uint32_t)k[3])<<24;
00791                 b += k[4];
00792                 b += ((uint32_t)k[5])<<8;
00793                 b += ((uint32_t)k[6])<<16;
00794                 b += ((uint32_t)k[7])<<24;
00795                 c += k[8];
00796                 c += ((uint32_t)k[9])<<8;
00797                 c += ((uint32_t)k[10])<<16;
00798                 c += ((uint32_t)k[11])<<24;
00799                 mix(a,b,c);
00800                 length -= 12;
00801                 k += 12;
00802             }
00803
00804             /*-------------------------------- last block: affect all 32 bits of (c) */
00805             switch(length) {                   /* all the case statements fall through */
00806                 case 12:
00807                     c+=((uint32_t)k[11])<<24;
00808                     /* no break */
00809                 case 11:
00810                     c+=((uint32_t)k[10])<<16;
00811                     /* no break */
00812                 case 10:
00813                     c+=((uint32_t)k[9])<<8;
00814                     /* no break */
00815                 case 9 :
00816                     c+=k[8];
00817                     /* no break */
00818                 case 8 :
00819                     b+=((uint32_t)k[7])<<24;
00820                     /* no break */
00821                 case 7 :
00822                     b+=((uint32_t)k[6])<<16;
00823                     /* no break */
00824                 case 6 :
00825                     b+=((uint32_t)k[5])<<8;
00826                     /* no break */
00827                 case 5 :
00828                     b+=k[4];
00829                     /* no break */
00830                 case 4 :
00831                     a+=((uint32_t)k[3])<<24;
00832                     /* no break */
00833                 case 3 :
00834                     a+=((uint32_t)k[2])<<16;
00835                     /* no break */
00836                 case 2 :
00837                     a+=((uint32_t)k[1])<<8;
00838                     /* no break */
```

```
00839                 case 1 :
00840                     a+=k[0];
00841                     break;
00842                 case 0 :
00843                     *pc=c;
00844                     *pb=b;
00845                     return;  /* zero length strings require no mixing */
00846             }
00847         }
00848
00849     final(a,b,c);
00850     *pc=c;
00851     *pb=b;
00852 }
```

12.27.3.4 uint32_t hashword (const uint32_t * *k*, size_t *length*, uint32_t *initval*)

hash a variable-length key into a 32-bit value (Big Endian)

```
--------------------------------------------------------------------
 This works on all machines.  To be useful, it requires
 -- that the key be an array of uint32_t's, and
 -- that the length be the number of uint32_t's in the key

 The function hashword() is identical to hashlittle() on little-endian
 machines, and identical to hashbig() on big-endian machines,
 except that the length has to be measured in uint32_ts rather than in
 bytes.  hashlittle() is more complicated than hashword() only because
 hashlittle() has to dance around fitting the key bytes into registers.
--------------------------------------------------------------------
```

Definition at line 182 of file lookup3.c.

References mix.

```
00185                             {       /* the previous hash, or an arbitrary value */
00186     uint32_t a,b,c;
00187
00188     /* Set up the internal state */
00189     a = b = c = 0xdeadbeef + (((uint32_t)length)<<2) + initval;
00190
00191     /*------------------------------------------------ handle most of the key */
00192     while (length > 3) {
00193         a += k[0];
00194         b += k[1];
00195         c += k[2];
00196         mix(a,b,c);
00197         length -= 3;
00198         k += 3;
00199     }
00200
00201     /*------------------------------------------- handle the last 3 uint32_t's */
00202     switch(length) {                   /* all the case statements fall through */
00203         case 3 :
00204             c+=k[2];
00205             /* no break */
00206         case 2 :
00207             b+=k[1];
00208             /* no break */
00209         case 1 :
00210             a+=k[0];
00211             final(a,b,c);
00212             /* no break */
00213         case 0:     /* case 0: nothing left to add */
00214             break;
00215     }
00216     /*------------------------------------------------------ report the result */
00217     return (c);
00218 }
```

12.27.3.5 void hashword2 (const uint32_t * *k*, size_t *length*, uint32_t * *pc*, uint32_t * *pb*)

same as hashword(), but take two seeds and return two 32-bit values

```
--------------------------------------------------------------------
hashword2() -- same as hashword(), but take two seeds and return two
32-bit values.  pc and pb must both be nonnull, and *pc and *pb must
both be initialized with seeds.  If you pass in (*pb)==0, the output
(*pc) will be the same as the return value from hashword().
--------------------------------------------------------------------
```

Definition at line 229 of file lookup3.c.

References mix.

```
00233                         {             /* IN: more seed OUT: secondary hash value */
00234     uint32_t a,b,c;
00235
00236     /* Set up the internal state */
00237     a = b = c = 0xdeadbeef + ((uint32_t)(length<<2)) + *pc;
00238     c += *pb;
00239
00240     /*------------------------------------------------ handle most of the key */
00241     while (length > 3) {
00242         a += k[0];
00243         b += k[1];
00244         c += k[2];
00245         mix(a,b,c);
00246         length -= 3;
00247         k += 3;
00248     }
00249
00250     /*------------------------------------------- handle the last 3 uint32_t's */
00251     switch(length) {                   /* all the case statements fall through */
00252         case 3 :
00253             c+=k[2];
00254             /* no break */
00255         case 2 :
00256             b+=k[1];
00257             /* no break */
00258         case 1 :
00259             a+=k[0];
00260             final(a,b,c);
00261             /* no break */
00262         case 0:     /* case 0: nothing left to add */
00263             break;
00264     }
00265     /*------------------------------------------------------ report the result */
00266     *pc=c;
00267     *pb=b;
00268 }
```

12.28 IPv6 Nat Mapping

Implementation of RFC6296.

Files

- file rfc6296.c

  Implementation of RFC6296.

Data Structures

- struct natmap

  RFC6296 Nat map.

Typedefs

- typedef struct natmap natmap

  Forward decleration of structure.

Functions

- void rfc6296_map (struct natmap *map, struct in6_addr *ipaddr, int out)

  Lookup and process a NAT transform as per RFC 6296.
- int rfc6296_map_add (char *intaddr, char *extaddr)

  Calculate and add a NAT map.
- void rfc6296_test (blist_cb callback, struct in6_addr *internal)

  Quick test function.

12.28.1 Detailed Description

Implementation of RFC6296.

12.28.2 Typedef Documentation

12.28.2.1 typedef struct natmap natmap

Forward decleration of structure.

Definition at line 189 of file dtsapp.h.

12.28.3 Function Documentation

12.28.3.1 void rfc6296_map (struct natmap * *map*, struct in6_addr * *ipaddr*, int *out*)

Lookup and process a NAT transform as per RFC 6296.

Parameters

map	Nat map structure to procees against.
ipaddr	Address to transform.
out	Set to non zero if ipaddr is internal and must be transformed to external.

Definition at line 62 of file rfc6296.c.

References natmap::adji, natmap::adjo, natmap::epre, natmap::ipre, and natmap::mask.

```
00062                                                                                   {
00063     uint16_t *addr_16 = (uint16_t *)&ipaddr->s6_addr;
00064     uint32_t calc;
00065     uint8_t cnt, *prefix, bitlen, bytelen;
00066     uint16_t adj;
00067 
00068     prefix = (out) ? map->epre : map->ipre;
00069     adj = (out) ? map->adjo : map->adji;
00070 
00071     if ((bitlen = map->mask % 8)) {
00072         bytelen = (map->mask - bitlen) / 8;
00073         bytelen++;
00074     } else {
00075         bytelen = map->mask / 8;
00076     }
00077 
00078     /*as per RFC we handle /48 and longer /48 changes are reflected in SN*/
00079     if ((bytelen == 6) && (~addr_16[3]) && (!bitlen)) {
00080         memcpy(&ipaddr->s6_addr, prefix, bytelen);
00081         calc = ntohs(addr_16[3]) + adj;
00082         addr_16[3] = htons((calc & 0xFFFF) + (calc >> 16));
00083         if (! ~addr_16[3]) {
00084             addr_16[3] = 0;
00085         }
00086     } else if ((bytelen > 6) && (bytelen < 15)) {
00087         /* find first non 0xFFFF word in lower 64 bits*/
00088         for(cnt = ((bytelen-1) >> 1) + 1; cnt < 8; cnt++) {
00089             if (! ~addr_16[cnt]) {
00090                 continue;
00091             }
00092             if (bitlen) {
00093                 ipaddr->s6_addr[bytelen-1] = prefix[bytelen-1] | (ipaddr->s6_addr[bytelen-1] & ((1 << (8 -
      bitlen)) -1));
00094             } else {
00095                 ipaddr->s6_addr[bytelen-1] = prefix[bytelen-1];
00096             }
00097             memcpy(&ipaddr->s6_addr, prefix, bytelen - 1);
00098             calc = ntohs(addr_16[cnt]) + adj;
00099             addr_16[cnt] = htons((calc & 0xFFFF) + (calc >> 16));
00100             if (! ~addr_16[cnt]) {
00101                 addr_16[cnt] = 0;
00102             }
00103             break;
00104         }
00105     }
00106 }
```

12.28.3.2 int rfc6296_map_add (char * *intaddr*, char * *extaddr*)

Calculate and add a NAT map.

Parameters

intaddr	Internal prefix/subnet.
extaddr	External prefix/subnet.

Definition at line 111 of file rfc6296.c.

References addtobucket(), natmap::adji, natmap::adjo, checksum(), create_bucketlist(), natmap::epre, natmap::ipre, natmap::mask, objalloc(), and objunref().

```
00111                                                       {
00112     struct natmap *map;
00113     uint16_t emask, imask, isum, esum, bytelen, bitlen;
00114     char inip[43], exip[43], *tmp2;
00115     struct in6_addr i6addr;
00116     uint32_t adj;
00117 
00118     strncpy(inip, intaddr, 43);
```

```
00119     if ((tmp2 = rindex(inip, '/'))) {
00120         tmp2[0] = '\0';
00121         tmp2++;
00122         imask = atoi(tmp2);
00123     } else {
00124         return (-1);
00125     }
00126 
00127     strncpy(exip, extaddr, 43);
00128     if ((tmp2 = rindex(exip, '/'))) {
00129         tmp2[0] = '\0';
00130         tmp2++;
00131         emask = atoi(tmp2);
00132     } else {
00133         return (-1);
00134     }
00135 
00136     map = objalloc(sizeof(*map), NULL);
00137     map->mask = (emask > imask) ? emask : imask;
00138 
00139     /*rfc says we must zero extend this is what we do here looking at each supplied len*/
00140     /*external range*/
00141     inet_pton(AF_INET6, exip, &i6addr);
00142     if ((bitlen = emask % 8)) {
00143         bytelen = (emask - bitlen) / 8;
00144         i6addr.s6_addr[bytelen] &= ~((1 << (8 - bitlen)) - 1);
00145         bytelen++;
00146     } else {
00147         bytelen = emask / 8;
00148     }
00149     memcpy(map->epre, &i6addr.s6_addr, bytelen);
00150 
00151     /*internal range*/
00152     inet_pton(AF_INET6, inip, &i6addr);
00153     if ((bitlen = imask % 8)) {
00154         bytelen = (imask - bitlen) / 8;
00155         i6addr.s6_addr[bytelen] &= ~((1 << (8 - bitlen)) - 1);
00156         bytelen++;
00157     } else {
00158         bytelen = imask / 8;
00159     }
00160     memcpy(map->ipre, &i6addr.s6_addr, bytelen);
00161 
00162     /*calculate the adjustments from checksums of prefixes*/
00163     if ((bitlen = map->mask % 8)) {
00164         bytelen = (map->mask - bitlen) / 8;
00165         bytelen++;
00166     } else {
00167         bytelen = map->mask / 8;
00168     }
00169     esum = ntohs(checksum(map->epre, bytelen));
00170     isum = ntohs(checksum(map->ipre, bytelen));
00171 
00172     /*outgoing transform*/
00173     adj = esum - isum;
00174     adj = (adj & 0xFFFF) + (adj >> 16);
00175     map->adjo = (uint16_t)adj;
00176 
00177     /*incoming transform*/
00178     adj = isum - esum;
00179     adj = (adj & 0xFFFF) + (adj >> 16);
00180     map->adji = (uint16_t)adj;
00181 
00182     if (!nptv6tbl && (!(nptv6tbl = create_bucketlist(5, nptv6_hash)))) {
00183         objunref(map);
00184         return (-1);
00185     }
00186     addtobucket(nptv6tbl, map);
00187     objunref(map);
00188 
00189     return (0);
00190 }
```

12.28.3.3 void rfc6296_test (blist_cb *callback*, struct in6_addr * *internal*)

Quick test function.

Run a callback against each entry in the table with the internal address as data.

Parameters

callback	Bucket list callback.
internal	Ip addr passed as data to the callback.

Definition at line 197 of file rfc6296.c.

References bucketlist_callback(), and objunref().

```
00197                                                                   {
00198     /*find and run map*/
00199     bucketlist_callback(nptv6tbl, callback, internal);
00200
00201     objunref(nptv6tbl);
00202 }
```

12.29 Windows Support

Support for building with mingw32 (Requires XP SP1+)

Files

- file winiface.cpp

  Various routines for supporting Windows also requires C++.

Data Structures

- struct ifinfo

  Data structure containing interface information.

Functions

- const char * inet_ntop (int af, const void *src, char *dest, socklen_t size)

  Win32 implementation of inet_ntop.
- struct ifinfo * get_ifinfo (const char *iface)

  Return interface info for a specified interface.

12.29.1 Detailed Description

Support for building with mingw32 (Requires XP SP1+)

12.29.2 Function Documentation

12.29.2.1 struct ifinfo* get_ifinfo (const char * *iface*)

Return interface info for a specified interface.

Parameters

iface	Interface name to return.

See Also

ifinfo

Returns

Reference to interface information structure

Definition at line 83 of file winiface.cpp.

References ifinfo::idx, ifinfo::ifaddr, ifinfo::ipv4addr, ifinfo::ipv6addr, objalloc(), score_ipv4(), score_ipv6(), and strlenzero().

Referenced by mcast_socket().

```
00083                                          {
00084     PIP_ADAPTER_ADDRESSES ainfo = NULL, cinfo;
00085     PIP_ADAPTER_UNICAST_ADDRESS pUnicast;
00086     struct sockaddr_storage *ss;
00087     char tmphn[NI_MAXHOST];
00088     char host4[NI_MAXHOST];
```

```
00089     char host6[NI_MAXHOST];
00090     int score4 = 0, score6 = 0, nscore;
00091     struct ifinfo *ifinf = NULL;
00092
00093     if (!(ainfo = get_adaptorinfo(15000, 3))) {
00094         return NULL;
00095     }
00096
00097     for(cinfo = ainfo; cinfo; cinfo = cinfo->Next) {
00098         if (strcmp(cinfo->AdapterName, iface)) {
00099             continue;
00100         }
00101
00102         if (!(ifinf = (struct ifinfo*)objalloc(sizeof(*ifinf),free_ifinfo))) {
00103             return NULL;
00104         }
00105
00106         ifinf->idx = (int)cinfo->IfIndex;
00107
00108         if (cinfo->PhysicalAddressLength == 6) {
00109             unsigned int i;
00110             char tmp[4];
00111             char tmp2[18] = "";
00112             for (i = 0; i < cinfo->PhysicalAddressLength; i++) {
00113                 if (i == (cinfo->PhysicalAddressLength - 1)) {
00114                     sprintf(tmp,"%.2X", (int)cinfo->PhysicalAddress[i]);
00115                 } else {
00116                     sprintf(tmp,"%.2X:", (int)cinfo->PhysicalAddress[i]);
00117                         }
00118                 strcat(tmp2, tmp);
00119             }
00120             ifinf->ifaddr = strdup(tmp2);
00121         } else {
00122             ifinf->ifaddr = NULL;
00123         }
00124
00125         for (pUnicast = cinfo->FirstUnicastAddress; pUnicast ;pUnicast = pUnicast->Next) {
00126             ss = (struct sockaddr_storage*)pUnicast->Address.lpSockaddr;
00127             switch(ss->ss_family) {
00128                 case AF_INET:
00129                     nscore = score_ipv4((struct sockaddr_in*)ss, tmphn, NI_MAXHOST);
00130                     if (score4 < nscore) {
00131                         score4 = nscore;
00132                         strcpy(host4, tmphn);
00133                     }
00134                     break;
00135                 case AF_INET6:
00136                     nscore = score_ipv6((struct sockaddr_in6*)ss, tmphn, NI_MAXHOST);
00137                     if (score6 < nscore) {
00138                         score6 = nscore;
00139                         strcpy(host6, tmphn);
00140                     }
00141                     break;
00142             }
00143         }
00144         ifinf->ipv4addr = (strlenzero(host4)) ? NULL : strdup(host4);
00145         ifinf->ipv6addr = (strlenzero(host6)) ? NULL : strdup(host6);
00146         break;
00147     }
00148
00149     if (ainfo) {
00150         free(ainfo);
00151     }
00152
00153     return ifinf;
00154 }
```

12.29.2.2 const char* inet_ntop (int *af*, const void * *src*, char * *dest*, socklen_t *size*)

Win32 implementation of inet_ntop.

Note

this is not a implemntation but a wrapper arround getnameinfo.

Parameters

af	Address family only AF_INET or AF_INET6 are supported.
src	A pointer to in_addr or in6_addr.
dest	A buffer to place the IP address in.
size	the length of the buffer.

Returns

Pointer to dest on success or NULL

Definition at line 43 of file winiface.cpp.

References sockstruct::sa, sockstruct::sa4, sockstruct::sa6, and sockstruct::ss.

Referenced by score_ipv4(), score_ipv6(), snprintf_pkt(), and sockaddr2ip().

```
00043                                                                                {
00044     union sockstruct sa;
00045     int res = 0;
00046     char serv[NI_MAXSERV];
00047 
00048     memset(&sa, 0, sizeof(sa));
00049     sa.ss.ss_family = af;
00050 
00051     switch(af) {
00052         case AF_INET:
00053             memcpy(&sa.sa4.sin_addr, src, sizeof(struct in_addr));
00054             res = getnameinfo(&sa.sa, sizeof(struct sockaddr_in), dest, size, serv, NI_MAXSERV,
      NI_NUMERICHOST | NI_NUMERICSERV);
00055             break;
00056         case AF_INET6:
00057             memcpy(&sa.sa6.sin6_addr, src, sizeof(struct in6_addr));
00058             res = getnameinfo(&sa.sa, sizeof(struct sockaddr_in6), dest, size, serv, NI_MAXSERV,
      NI_NUMERICHOST | NI_NUMERICSERV);
00059             break;
00060     }
00061     return (!res) ? dest : NULL;
00062 }
```

12.30 Distrotech Application Library (Todo)

Modules not completely documented.

Modules

- Linux Netfilter

  Interface to linux netfilter.

12.30.1 Detailed Description

Modules not completely documented.

12.31 Linux Netfilter

Interface to linux netfilter.

Modules

- Connection Tracking

  Interface to linux netfilter connection tracking.
- Queue interface

  Interface to linux netfilter queue interface.

12.31.1 Detailed Description

Interface to linux netfilter.

12.32 Connection Tracking

Interface to linux netfilter connection tracking.

Files

- file nf_ctrack.c

  linux Netfilter Connection Tracking

Data Structures

- struct **nfct_struct**

Typedefs

- typedef struct nfct_struct nfct_struct

  Forward decleration of structure.

Enumerations

- enum NF_CTRACK_FLAGS { NFCTRACK_DONE = 1 << 0 }

  Netfilter Ctrack Flags.

Functions

- uint8_t nf_ctrack_init (void)
- struct nf_conntrack * nf_ctrack_buildct (uint8_t *pkt)
- uint8_t nf_ctrack_delete (uint8_t *pkt)
- uint8_t nf_ctrack_nat (uint8_t *pkt, uint32_t addr, uint16_t port, uint8_t dnat)
- void nf_ctrack_dump (void)
- struct nfct_struct * nf_ctrack_trace (void)
- void nf_ctrack_endtrace (struct nfct_struct *nfct)
- void nf_ctrack_close (void)

12.32.1 Detailed Description

Interface to linux netfilter connection tracking.

12.32.2 Typedef Documentation

12.32.2.1 typedef struct nfct_struct nfct_struct

Forward decleration of structure.

Definition at line 205 of file dtsapp.h.

12.32.3 Enumeration Type Documentation

12.32.3.1 enum NF_CTRACK_FLAGS

Netfilter Ctrack Flags.

Enumerator

NFCTRACK_DONE

Definition at line 44 of file nf_ctrack.c.

```
00044                       {
00045     NFCTRACK_DONE     = 1 << 0
00046 };
```

12.32.4 Function Documentation

12.32.4.1 struct nf_conntrack* nf_ctrack_buildct (uint8_t * *pkt*)

Definition at line 97 of file nf_ctrack.c.

Referenced by nf_ctrack_delete(), and nf_ctrack_nat().

```
00097                                                                   {
00098     struct nf_conntrack *ct;
00099     struct iphdr *ip = (struct iphdr *)pkt;
00100     union l4hdr *l4 = (union l4hdr *)(pkt + (ip->ihl * 4));
00101
00102     if (!(ct = nfct_new())) {
00103         return (NULL);
00104     };
00105
00106     /*Build tuple*/
00107     nfct_set_attr_u8(ct, ATTR_L3PROTO, PF_INET);
00108     nfct_set_attr_u32(ct, ATTR_IPV4_SRC, ip->saddr);
00109     nfct_set_attr_u32(ct, ATTR_IPV4_DST, ip->daddr);
00110     nfct_set_attr_u8(ct, ATTR_L4PROTO, ip->protocol);
00111     switch(ip->protocol) {
00112         case IPPROTO_TCP:
00113             nfct_set_attr_u16(ct, ATTR_PORT_SRC, l4->tcp.source);
00114             nfct_set_attr_u16(ct, ATTR_PORT_DST, l4->tcp.dest);
00115             break;
00116         case IPPROTO_UDP:
00117             nfct_set_attr_u16(ct, ATTR_PORT_SRC, l4->udp.source);
00118             nfct_set_attr_u16(ct, ATTR_PORT_DST, l4->udp.dest);
00119             break;
00120         case IPPROTO_ICMP:
00121             nfct_set_attr_u8(ct, ATTR_ICMP_TYPE, l4->icmp.type);
00122             nfct_set_attr_u8(ct, ATTR_ICMP_CODE, l4->icmp.code);
00123             nfct_set_attr_u16(ct, ATTR_ICMP_ID, l4->icmp.un.echo.id);
00124             /* no break */
00125         default
00126                 :
00127             break;
00128     };
00129
00130     return (ct);
00131 }
```

12.32.4.2 void nf_ctrack_close (void)

Definition at line 285 of file nf_ctrack.c.

References objunref().

Referenced by nf_ctrack_delete(), nf_ctrack_dump(), and nf_ctrack_nat().

```
00285                                   {
00286     if (ctrack) {
```

```
00287         objunref(ctrack);
00288     }
00289     ctrack = NULL;
00290 }
```

12.32.4.3 uint8_t nf_ctrack_delete (uint8_t * *pkt*)

Definition at line 133 of file nf_ctrack.c.

References nf_ctrack_buildct(), nf_ctrack_close(), nf_ctrack_init(), objlock(), and objunlock().

```
00133                                                {
00134     struct nf_conntrack *ct;
00135     uint8_t unref = 0;
00136     uint8_t ret = 0;
00137
00138     if (!ctrack) {
00139         if (nf_ctrack_init()) {
00140             return (-1);
00141         }
00142         unref = 1;
00143     }
00144
00145     ct = nf_ctrack_buildct(pkt);
00146     objlock(ctrack);
00147     if (nfct_query(ctrack->nfct, NFCT_Q_DESTROY, ct) < 0) {
00148         ret = -1;
00149     }
00150     objunlock(ctrack);
00151     nfct_destroy(ct);
00152
00153     if (unref) {
00154         nf_ctrack_close();
00155     }
00156
00157     return (ret);
00158 }
```

12.32.4.4 void nf_ctrack_dump (void)

Definition at line 204 of file nf_ctrack.c.

References nf_ctrack_close(), nf_ctrack_init(), objlock(), and objunlock().

```
00204                                {
00205     uint32_t family = PF_INET;
00206     uint8_t unref = 0;
00207
00208     if (!ctrack) {
00209         if (nf_ctrack_init()) {
00210             return;
00211         }
00212         unref = 1;
00213     }
00214
00215     objlock(ctrack);
00216     nfct_callback_register(ctrack->nfct, NFCT_T_ALL, nfct_cb, NULL);
00217     nfct_query(ctrack->nfct, NFCT_Q_DUMP, &family);
00218     nfct_callback_unregister(ctrack->nfct);
00219     objunlock(ctrack);
00220
00221     if (unref) {
00222         nf_ctrack_close();
00223     }
00224 }
```

12.32.4.5 void nf_ctrack_endtrace (struct nfct_struct * *nfct*)

Definition at line 278 of file nf_ctrack.c.

References NFCTRACK_DONE, objunref(), and setflag.

```
00278                                                   {
00279     if (nfct) {
00280         setflag(nfct, NFCTRACK_DONE);
00281     }
00282     objunref(nfct);
00283 }
```

12.32.4.6 uint8_t nf_ctrack_init (void)

Definition at line 90 of file nf_ctrack.c.

Referenced by nf_ctrack_delete(), nf_ctrack_dump(), and nf_ctrack_nat().

```
00090                                   {
00091     if (!ctrack && !(ctrack = nf_ctrack_alloc(CONNTRACK, 0))) {
00092         return (-1);
00093     }
00094     return (0);
00095 }
```

12.32.4.7 uint8_t nf_ctrack_nat (uint8_t * *pkt*, uint32_t *addr*, uint16_t *port*, uint8_t *dnat*)

Definition at line 160 of file nf_ctrack.c.

References nf_ctrack_buildct(), nf_ctrack_close(), nf_ctrack_init(), objlock(), and objunlock().

```
00160                                                                                                   {
00161     struct iphdr *ip = (struct iphdr *)pkt;
00162     struct nf_conntrack *ct;
00163     uint8_t unref = 0;
00164     uint8_t ret = 0;
00165
00166     if (!ctrack) {
00167         if (nf_ctrack_init()) {
00168             return (-1);
00169         }
00170         unref = 1;
00171     }
00172
00173     ct = nf_ctrack_buildct(pkt);
00174     nfct_setobjopt(ct, NFCT_SOPT_SETUP_REPLY);
00175
00176     nfct_set_attr_u32(ct, ATTR_TIMEOUT, 120);
00177     nfct_set_attr_u32(ct, (dnat) ? ATTR_DNAT_IPV4 : ATTR_SNAT_IPV4, addr);
00178
00179     switch(ip->protocol) {
00180         case IPPROTO_TCP:
00181             nfct_set_attr_u8(ct, ATTR_TCP_STATE, TCP_CONNTRACK_ESTABLISHED);
00182             /* no break */
00183         case IPPROTO_UDP:
00184             if (port) {
00185                 nfct_set_attr_u16(ct, (dnat) ? ATTR_DNAT_PORT : ATTR_SNAT_PORT, port);
00186             }
00187             break;
00188     }
00189
00190     objlock(ctrack);
00191     if (nfct_query(ctrack->nfct, NFCT_Q_CREATE_UPDATE, ct) < 0) {
00192         ret = -1;
00193     }
00194     objunlock(ctrack);
00195     nfct_destroy(ct);
00196
00197     if (unref) {
00198         nf_ctrack_close();
00199     }
00200
00201     return (ret);
00202 }
```

12.32.4.8 struct nfct_struct* nf_ctrack_trace (void)

Definition at line 261 of file nf_ctrack.c.

References framework_mkthread(), objunref(), and THREAD_OPTION_RETURN.

```
00261                                           {
00262     struct nfct_struct *nfct;
00263     void *thr;
00264
00265     if (!(nfct = nf_ctrack_alloc(CONNTRACK, NFCT_ALL_CT_GROUPS))) {
00266         return (NULL);
00267     }
00268
00269     if (!(thr = framework_mkthread(nf_ctrack_trace_th, NULL, NULL, nfct,
      THREAD_OPTION_RETURN))) {
00270         objunref(nfct);
00271         return NULL;
00272     }
00273     objunref(thr);
00274
00275     return (nfct);
00276 }
```

12.33 Queue interface

Interface to linux netfilter queue interface.

Files

- file nf_queue.c

  Linux netfilter queue interface.

Data Structures

- struct nfq_struct
- struct nfq_queue
- struct **nfq_list**

Typedefs

- typedef struct nfq_queue nfq_queue

  Forward decleration of structure.
- typedef struct nfq_data nfq_data

  Forward decleration of structure.
- typedef struct nfqnl_msg_packet_hdr nfqnl_msg_packet_hdr

  Forward decleration of structure.
- typedef uint32_t(∗ nfqueue_cb)(struct nfq_data ∗, struct nfqnl_msg_packet_hdr ∗, char ∗, uint32_t, void ∗, uint32_t ∗, void ∗∗)

Enumerations

- enum NF_QUEUE_FLAGS { NFQUEUE_DONE = 1 << 0 }

Functions

- struct nfq_queue ∗ nfqueue_attach (uint16_t pf, uint16_t num, uint8_t mode, uint32_t range, nfqueue_cb cb, void ∗data)
- uint16_t snprintf_pkt (struct nfq_data ∗tb, struct nfqnl_msg_packet_hdr ∗ph, uint8_t ∗pkt, char ∗buff, uint16_t len)

12.33.1 Detailed Description

Interface to linux netfilter queue interface.

12.33.2 Typedef Documentation

12.33.2.1 typedef struct nfq_data nfq_data

Forward decleration of structure.

Definition at line 201 of file dtsapp.h.

12.33.2.2 typedef struct nfq_queue nfq_queue

Forward decleration of structure.

Definition at line 197 of file dtsapp.h.

12.33.2.3 typedef struct nfqnl_msg_packet_hdr nfqnl_msg_packet_hdr

Forward decleration of structure.

Definition at line 209 of file dtsapp.h.

12.33.2.4 typedef uint32_t(∗ nfqueue_cb)(struct nfq_data ∗, struct nfqnl_msg_packet_hdr ∗, char ∗, uint32_t, void ∗, uint32_t ∗, void ∗∗)

Definition at line 300 of file dtsapp.h.

12.33.3 Enumeration Type Documentation

12.33.3.1 enum NF_QUEUE_FLAGS

Enumerator

NFQUEUE_DONE

Definition at line 43 of file nf_queue.c.

```
00043                       {
00044     NFQUEUE_DONE      = 1 << 0
00045 };
```

12.33.4 Function Documentation

12.33.4.1 struct nfq_queue∗ nfqueue_attach (uint16_t *pf,* uint16_t *num,* uint8_t *mode,* uint32_t *range,* nfqueue_cb *cb,* void ∗ *data*)

Definition at line 231 of file nf_queue.c.

References bucket_list_find_key(), nfq_queue::cb, nfq_queue::data, nfq_struct::h, nfq_queue::nfq, objalloc(), objlock(), objunlock(), objunref(), and nfq_queue::qh.

```
00231
                    {
00232     struct nfq_queue *nfq_q;
00233
00234     if (!(nfq_q = objalloc(sizeof(*nfq_q), nfqueue_close_q))) {
00235         return (NULL);
00236     }
00237
00238     objlock(nfqueues);
00239     if (!(nfqueues && (nfq_q->nfq = bucket_list_find_key(nfqueues->queues, &pf))) &&
00240             !(nfq_q->nfq || (nfq_q->nfq = nfqueue_init(pf)))) {
00241         objunlock(nfqueues);
00242         objunref(nfq_q);
00243         return (NULL);
00244     }
00245     objunlock(nfqueues);
00246
00247     if (!(nfq_q->qh = nfq_create_queue(nfq_q->nfq->h, num, &nfqueue_callback, nfq_q))) {
00248         objunref(nfq_q);
00249         return (NULL);
00250     }
00251
00252     if (cb) {
00253         nfq_q->cb = cb;
00254     }
```

```
00255
00256     if (data) {
00257         nfq_q->data = data;
00258     }
00259
00260     nfq_set_mode(nfq_q->qh, mode, range);
00261
00262     return (nfq_q);
00263 }
```

12.33.4.2 uint16_t snprintf_pkt (struct nfq_data * *tb*, struct nfqnl_msg_packet_hdr * *ph*, uint8_t * *pkt*, char * *buff*, uint16_t *len*)

Definition at line 265 of file nf_queue.c.

References inet_ntop().

```
00265
                    {
00266     struct iphdr *ip = (struct iphdr *)pkt;
00267     char *tmp = buff;
00268     uint32_t id, mark, ifi;
00269     uint16_t tlen, left = len;
00270     char saddr[INET_ADDRSTRLEN], daddr[INET_ADDRSTRLEN];
00271
00272     if (ph) {
00273         id = ntohl(ph->packet_id);
00274         snprintf(tmp, left, "hw_protocol=0x%04x hook=%u id=%u ",
00275                  ntohs(ph->hw_protocol), ph->hook, id);
00276         tlen = strlen(tmp);
00277         tmp += tlen;
00278         left -= tlen;
00279     }
00280
00281     if ((mark = nfq_get_nfmark(tb))) {
00282         snprintf(tmp, left, "mark=%u ", mark);
00283         tlen = strlen(tmp);
00284         tmp += tlen;
00285         left -= tlen;
00286     }
00287
00288     if ((ifi = nfq_get_indev(tb))) {
00289         snprintf(tmp, left, "indev=%u ", ifi);
00290         tlen = strlen(tmp);
00291         tmp += tlen;
00292         left -= tlen;
00293     }
00294
00295     if ((ifi = nfq_get_outdev(tb))) {
00296         snprintf(tmp, left, "outdev=%u ", ifi);
00297         tlen = strlen(tmp);
00298         tmp += tlen;
00299         left -= tlen;
00300     }
00301
00302     if (pkt && (ip->version == 4)) {
00303         union l4hdr *l4 = (union l4hdr *)(pkt + (ip->ihl*4));
00304
00305         inet_ntop(AF_INET, &ip->saddr, saddr, INET_ADDRSTRLEN);
00306         inet_ntop(AF_INET, &ip->daddr, daddr, INET_ADDRSTRLEN);
00307
00308         snprintf(tmp, left, "src=%s dst=%s proto=%i ", saddr, daddr, ip->protocol);
00309         tlen = strlen(tmp);
00310         tmp += tlen;
00311         left -= tlen;
00312
00313         switch(ip->protocol) {
00314             case IPPROTO_TCP:
00315                 snprintf(tmp, left, "sport=%i dport=%i ", ntohs(l4->tcp.source), ntohs(l4->tcp.dest));
00316                 break;
00317             case IPPROTO_UDP:
00318                 snprintf(tmp, left, "sport=%i dport=%i ", ntohs(l4->udp.source), ntohs(l4->udp.dest));
00319                 break;
00320             case IPPROTO_ICMP:
00321                 snprintf(tmp, left, "type=%i code=%i id=%i ", l4->icmp.type, l4->icmp.code, ntohs(l4->icmp.
    un.echo.id));
00322                 break;
00323         }
00324         tlen = strlen(tmp);
00325         tmp += tlen;
00326         left -= tlen;
```

```
00327     }
00328
00329     return (len - left);
00330 }
```

Chapter 13

Data Structure Documentation

13.1 basic_auth Struct Reference

Basic authentification structure.

`#include <dtsapp.h>`

Data Fields

- const char * user
  Username.
- const char * passwd
  Password.

13.1.1 Detailed Description

Basic authentification structure.

Definition at line 824 of file dtsapp.h.

13.1.2 Field Documentation

13.1.2.1 const char* basic_auth::passwd

Password.

Definition at line 828 of file dtsapp.h.

Referenced by curl_newauth().

13.1.2.2 const char* basic_auth::user

Username.

Definition at line 826 of file dtsapp.h.

Referenced by curl_newauth().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.2 blist_obj Struct Reference

Entry in a bucket list.

Data Fields

- int32_t hash

  Hash value calculated from the data.
- struct blist_obj * next

  Next entry in the bucket.
- struct blist_obj * prev

  Previous entry in the bucket.
- struct ref_obj * data

  Reference to data held.

13.2.1 Detailed Description

Entry in a bucket list.

Definition at line 61 of file refobj.c.

13.2.2 Field Documentation

13.2.2.1 struct ref_obj* blist_obj::data

Reference to data held.

Definition at line 70 of file refobj.c.

Referenced by addtobucket(), bucket_list_find_key(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

13.2.2.2 int32_t blist_obj::hash

Hash value calculated from the data.

Warning

this should not change during the life of this object

Definition at line 64 of file refobj.c.

Referenced by addtobucket(), bucket_list_find_key(), init_bucket_loop(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

13.2.2.3 struct blist_obj* blist_obj::next

Next entry in the bucket.

Definition at line 66 of file refobj.c.

Referenced by addtobucket(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

13.2.2.4 struct blist_obj* blist_obj::prev

Previous entry in the bucket.

Definition at line 68 of file refobj.c.

Referenced by addtobucket(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

The documentation for this struct was generated from the following file:

- src/refobj.c

13.3 bucket_list Struct Reference

Bucket list, hold hashed objects in buckets.

Data Fields

- unsigned short bucketbits

  number of buckets 2^n
- size_t count

  Number of items held.
- blisthash hash_func

  Hash function called to calculate the hash and thus the bucket its placed in.
- struct blist_obj ** list

  Array of blist_obj one per bucket ie 2^bucketbits.
- pthread_mutex_t * locks

  Array of locks one per bucket.
- size_t * version

  version of the bucket to detect changes during iteration (loop)

13.3.1 Detailed Description

Bucket list, hold hashed objects in buckets.

Definition at line 75 of file refobj.c.

13.3.2 Field Documentation

13.3.2.1 unsigned short bucket_list::bucketbits

number of buckets 2^n

Definition at line 77 of file refobj.c.

Referenced by addtobucket(), bucket_list_find_key(), create_bucketlist(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

13.3.2.2 size_t bucket_list::count

Number of items held.

Definition at line 79 of file refobj.c.

Referenced by addtobucket(), bucket_list_cnt(), remove_bucket_item(), and remove_bucket_loop().

13.3.2.3 blisthash bucket_list::hash_func

Hash function called to calculate the hash and thus the bucket its placed in.

Definition at line 81 of file refobj.c.

13.3.2.4 struct blist_obj** bucket_list::list

Array of blist_obj one per bucket ie 2^bucketbits.

Definition at line 83 of file refobj.c.

Referenced by addtobucket(), bucket_list_find_key(), init_bucket_loop(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

13.3.2.5 pthread_mutex_t* bucket_list::locks

Array of locks one per bucket.

Definition at line 85 of file refobj.c.

Referenced by addtobucket(), bucket_list_find_key(), init_bucket_loop(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

13.3.2.6 size_t* bucket_list::version

version of the bucket to detect changes during iteration (loop)

Definition at line 87 of file refobj.c.

Referenced by addtobucket(), init_bucket_loop(), next_bucket_loop(), remove_bucket_item(), and remove_bucket_loop().

The documentation for this struct was generated from the following file:

- src/refobj.c

13.4 bucket_loop Struct Reference

Bucket iterator.

Data Fields

- struct bucket_list * blist
  Referenece to the bucket been itereated.
- unsigned short bucket
  Active bucket as determined by hash.
- size_t version
  Our version check this with blist to determine if we must rewined and fast forward.
- uint32_t head_hash
  Hash of head if we need to comeback.
- uint32_t cur_hash
  Hash of cur if we need to comeback.
- struct blist_obj * head
  Current bucket.
- struct blist_obj * cur
  Current item.

13.4.1 Detailed Description

Bucket iterator.

buckets are more complex than linked lists to loop through them we will use a structure that holds a reference to the bucket and head it needs to be initialised and destroyed

Definition at line 97 of file refobj.c.

13.4.2 Field Documentation

13.4.2.1 struct bucket_list* bucket_loop::blist

Referenece to the bucket been itereated.

Definition at line 99 of file refobj.c.

Referenced by init_bucket_loop(), next_bucket_loop(), and remove_bucket_loop().

13.4.2.2 unsigned short bucket_loop::bucket

Active bucket as determined by hash.

Definition at line 101 of file refobj.c.

Referenced by init_bucket_loop(), next_bucket_loop(), and remove_bucket_loop().

13.4.2.3 struct blist_obj* bucket_loop::cur

Current item.

Definition at line 112 of file refobj.c.

Referenced by next_bucket_loop(), and remove_bucket_loop().

13.4.2.4 uint32_t bucket_loop::cur_hash

Hash of cur if we need to comeback.

Definition at line 108 of file refobj.c.

Referenced by next_bucket_loop(), and remove_bucket_loop().

13.4.2.5 struct blist_obj* bucket_loop::head

Current bucket.

Definition at line 110 of file refobj.c.

Referenced by init_bucket_loop(), and next_bucket_loop().

13.4.2.6 uint32_t bucket_loop::head_hash

Hash of head if we need to comeback.

Definition at line 106 of file refobj.c.

Referenced by init_bucket_loop(), and next_bucket_loop().

13.4.2.7 size_t bucket_loop::version

Our version check this with blist to determine if we must rewined and fast forward.

Definition at line 104 of file refobj.c.

Referenced by init_bucket_loop(), next_bucket_loop(), and remove_bucket_loop().

The documentation for this struct was generated from the following file:

- src/refobj.c

13.5 config_category Struct Reference

Configuration file category.

Data Fields

- const char * name
  Category name.
- struct bucket_list * entries
  Entries in category.

13.5.1 Detailed Description

Configuration file category.

Definition at line 32 of file config.c.

13.5.2 Field Documentation

13.5.2.1 struct bucket_list* config_category::entries

Entries in category.

Definition at line 36 of file config.c.

Referenced by get_category_next(), and get_config_category().

13.5.2.2 const char* config_category::name

Category name.

Definition at line 34 of file config.c.

Referenced by get_category_next().

The documentation for this struct was generated from the following file:

- src/config.c

13.6 config_entry Struct Reference

Configuration category entry.

```
#include <dtsapp.h>
```

Data Fields

- const char * item
- const char * value

13.6.1 Detailed Description

Configuration category entry.

Definition at line 155 of file dtsapp.h.

13.6.2 Field Documentation

13.6.2.1 const char* config_entry::item

@ brief Item name

Definition at line 157 of file dtsapp.h.

13.6.2.2 const char* config_entry::value

@ brief Item value

Definition at line 159 of file dtsapp.h.

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.7 config_file Struct Reference

Config file.

Data Fields

- const char * filename
  Filename.
- const char * filepath
  File path.
- struct bucket_list * cat
  Categories.

13.7.1 Detailed Description

Config file.

Definition at line 40 of file config.c.

13.7.2 Field Documentation

13.7.2.1 struct bucket_list* config_file::cat

Categories.

Definition at line 46 of file config.c.

Referenced by get_config_file(), and process_config().

13.7.2.2 const char* config_file::filename

Filename.

Definition at line 42 of file config.c.

13.7.2.3 const char* config_file::filepath

File path.

Definition at line 44 of file config.c.

Referenced by process_config().

The documentation for this struct was generated from the following file:

- src/config.c

13.8 curl_post Struct Reference

HTTP post data structure.

Data Fields

- struct curl_httppost * first
  First item in the list.
- struct curl_httppost * last
  Last item in the list.

13.8.1 Detailed Description

HTTP post data structure.

Definition at line 40 of file curl.c.

13.8.2 Field Documentation

13.8.2.1 struct curl_httppost* curl_post::first

First item in the list.

Definition at line 42 of file curl.c.

Referenced by curl_newpost(), and curl_postitem().

13.8.2.2 struct curl_httppost* curl_post::last

Last item in the list.

Definition at line 44 of file curl.c.

Referenced by curl_newpost(), and curl_postitem().

The documentation for this struct was generated from the following file:

- src/curl.c

13.9 curlbuf Struct Reference

Buffer containing the result of a curl transaction.

`#include <dtsapp.h>`

Data Fields

- uint8_t * header
  Header buffer.
- uint8_t * body
  Body buffer.
- char * c_type
  Mime Type.
- size_t hsize
  Header size.
- size_t bsize
  Body size.

13.9.1 Detailed Description

Buffer containing the result of a curl transaction.

Definition at line 832 of file dtsapp.h.

13.9.2 Field Documentation

13.9.2.1 uint8_t* curlbuf::body

Body buffer.

Definition at line 836 of file dtsapp.h.

Referenced by curl_buf2xml(), and curl_ungzip().

13.9.2.2 size_t curlbuf::bsize

Body size.

Definition at line 842 of file dtsapp.h.

Referenced by curl_buf2xml(), and curl_ungzip().

13.9.2.3 char* curlbuf::c_type

Mime Type.

Definition at line 838 of file dtsapp.h.

Referenced by curl_buf2xml().

13.9.2.4 uint8_t* curlbuf::header

Header buffer.

Definition at line 834 of file dtsapp.h.

13.9.2.5 size_t curlbuf::hsize

Header size.

Definition at line 840 of file dtsapp.h.

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.10 framework_core Struct Reference

Application framework data.

`#include <dtsapp.h>`

Data Fields

- const char * developer
  Developer/Copyright holder.
- const char * email
  Email address of copyright holder.
- const char * www
  URL displayed (use full URL ie with `http://`)
- const char * runfile
  File to write PID too and lock.
- const char * progname
  Detailed application name.
- int year
  Copyright year.
- int flock
  if there is a file locked this is the FD that will be unlocked and unlinked
- struct sigaction * sa
  sigaction structure allocated on execution
- syssighandler sig_handler
  Signal handler to pass signals too.
- int flags
  Application Options.

13.10.1 Detailed Description

Application framework data.

See Also

framework_mkcore()
framework_init()
FRAMEWORK_MAIN()

Definition at line 326 of file dtsapp.h.

13.10.2 Field Documentation

13.10.2.1 const char* framework_core::developer

Developer/Copyright holder.

Definition at line 328 of file dtsapp.h.

Referenced by framework_init(), and framework_mkcore().

13.10.2.2 const char* framework_core::email

Email address of copyright holder.

Definition at line 330 of file dtsapp.h.

Referenced by framework_init(), and framework_mkcore().

13.10.2.3 int framework_core::flags

Application Options.

See Also

application_flags

Definition at line 348 of file dtsapp.h.

Referenced by daemonize(), framework_init(), and framework_mkcore().

13.10.2.4 int framework_core::flock

if there is a file locked this is the FD that will be unlocked and unlinked

Definition at line 340 of file dtsapp.h.

Referenced by daemonize(), framework_init(), and lockpidfile().

13.10.2.5 const char* framework_core::progname

Detailed application name.

Definition at line 336 of file dtsapp.h.

Referenced by framework_init(), and framework_mkcore().

13.10.2.6 const char* framework_core::runfile

File to write PID too and lock.

Definition at line 334 of file dtsapp.h.

Referenced by daemonize(), framework_init(), and framework_mkcore().

13.10.2.7 struct sigaction* framework_core::sa

sigaction structure allocated on execution

Definition at line 342 of file dtsapp.h.

Referenced by framework_init(), and framework_mkcore().

13.10.2.8 syssighandler framework_core::sig_handler

Signal handler to pass signals too.

Note

The application framework installs a signal handler but will pass calls to this as a callback

Definition at line 345 of file dtsapp.h.

Referenced by framework_mkcore().

13.10.2.9 const char* framework_core::www

URL displayed (use full URL ie with `http://`)

Definition at line 332 of file dtsapp.h.

Referenced by framework_init(), and framework_mkcore().

13.10.2.10 int framework_core::year

Copyright year.

Definition at line 338 of file dtsapp.h.

Referenced by framework_init(), and framework_mkcore().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.11 fwsocket Struct Reference

Socket data structure.

`#include <dtsapp.h>`

Data Fields

- int sock
  Socket FD.
- int proto
  Socket protocol.
- int type
  Socket type.
- enum sock_flags flags
  Socket control flags.
- union sockstruct addr
  system socket data structure.
- struct ssldata * ssl
  SSL structure for encryption.
- struct fwsocket * parent
  Parent socket if we connected to a server and were spawned.
- struct bucket_list * children
  We are the parent this is a list of spawn.

13.11.1 Detailed Description

Socket data structure.

Examples:

socket.c.

Definition at line 131 of file dtsapp.h.

13.11.2 Field Documentation

13.11.2.1 union sockstruct fwsocket::addr

system socket data structure.

See Also

sockstruct

Definition at line 143 of file dtsapp.h.

Referenced by accept_socket(), dtls_listenssl(), mcast_socket(), socketwrite_d(), and unixsocket_client().

13.11.2.2 struct bucket_list* fwsocket::children

We are the parent this is a list of spawn.

Definition at line 150 of file dtsapp.h.

Referenced by socketserver().

13.11.2.3 enum sock_flags fwsocket::flags

Socket control flags.

See Also

sock_flags

Definition at line 140 of file dtsapp.h.

Referenced by mcast_socket(), socketread_d(), socketserver(), socketwrite_d(), and unixsocket_client().

13.11.2.4 struct fwsocket* fwsocket::parent

Parent socket if we connected to a server and were spawned.

Definition at line 148 of file dtsapp.h.

13.11.2.5 int fwsocket::proto

Socket protocol.

Definition at line 135 of file dtsapp.h.

Referenced by accept_socket(), dtls_listenssl(), and make_socket().

13.11.2.6 int fwsocket::sock

Socket FD.

Examples:

socket.c.

Definition at line 133 of file dtsapp.h.

Referenced by accept_socket(), client_func(), dtls_listenssl(), make_socket(), mcast_socket(), server_func(), socketread_d(), socketwrite_d(), and unixsocket_client().

13.11.2.7 struct ssldata* fwsocket::ssl

SSL structure for encryption.

See Also

SSL socket support

Definition at line 146 of file dtsapp.h.

Referenced by accept_socket(), dtls_listenssl(), dtlshandltimeout(), dtlstimeout(), dtsl_serveropts(), make_socket(), socketread_d(), socketserver(), socketwrite_d(), startsslclient(), and tlsaccept().

13.11.2.8 int fwsocket::type

Socket type.

Definition at line 137 of file dtsapp.h.

Referenced by accept_socket(), dtls_listenssl(), make_socket(), socketread_d(), socketserver(), socketwrite_d(), and startsslclient().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.12 ifinfo Struct Reference

Data structure containing interface information.

`#include <dtsapp.h>`

Data Fields

- int idx

  Interface index required for at least IPv6 multicast support.
- const char * ifaddr

  MAC address of interface.
- const char * ipv4addr

  IPv4 address priorotised by Routed/Reserved/Zeroconf.
- const char * ipv6addr

  IPv6 address priorised by Local/6in4.

13.12.1 Detailed Description

Data structure containing interface information.

Note

This is specific to Windows XP SP1+

Definition at line 176 of file dtsapp.h.

13.12.2 Field Documentation

13.12.2.1 int ifinfo::idx

Interface index required for at least IPv6 multicast support.

Definition at line 178 of file dtsapp.h.

Referenced by get_ifinfo(), and mcast_socket().

13.12.2.2 const char* ifinfo::ifaddr

MAC address of interface.

Definition at line 180 of file dtsapp.h.

Referenced by get_ifinfo().

13.12.2.3 const char* ifinfo::ipv4addr

IPv4 address priorotised by Routed/Reserved/Zeroconf.

Definition at line 182 of file dtsapp.h.

Referenced by get_ifinfo(), and mcast_socket().

13.12.2.4 const char* ifinfo::ipv6addr

IPv6 address priorised by Local/6in4.

Definition at line 184 of file dtsapp.h.

Referenced by get_ifinfo(), and mcast_socket().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.13 ipaddr_req Struct Reference

IP Netlink IP addr request.

Data Fields

- struct nlmsghdr n

  Netlink message header.
- struct ifaddrmsg i

Interface addr message.

- char buf [1024]

  Request buffer.

13.13.1 Detailed Description

IP Netlink IP addr request.

Definition at line 95 of file interface.c.

13.13.2 Field Documentation

13.13.2.1 char ipaddr_req::buf[1024]

Request buffer.

Definition at line 101 of file interface.c.

13.13.2.2 struct ifaddrmsg ipaddr_req::i

Interface addr message.

Definition at line 99 of file interface.c.

Referenced by set_interface_ipaddr().

13.13.2.3 struct nlmsghdr ipaddr_req::n

Netlink message header.

Definition at line 97 of file interface.c.

Referenced by set_interface_ipaddr().

The documentation for this struct was generated from the following file:

- src/interface.c

13.14 iplink_req Struct Reference

IP Netlink request.

Data Fields

- struct nlmsghdr n

  Netlink message header.

- struct ifinfomsg i

  Interface info message.

- char buf [1024]

  Request buffer.

13.14.1 Detailed Description

IP Netlink request.

Definition at line 85 of file interface.c.

13.14.2 Field Documentation

13.14.2.1 char iplink_req::buf[1024]

Request buffer.

Definition at line 91 of file interface.c.

13.14.2.2 struct ifinfomsg iplink_req::i

Interface info message.

Definition at line 89 of file interface.c.

Referenced by set_interface_addr(), set_interface_flags(), and set_interface_name().

13.14.2.3 struct nlmsghdr iplink_req::n

Netlink message header.

Definition at line 87 of file interface.c.

Referenced by create_kernmac(), create_kernvlan(), set_interface_addr(), set_interface_flags(), and set_interface_name().

The documentation for this struct was generated from the following file:

- src/interface.c

13.15 ldap_add Struct Reference

LDAP Add structure.

Data Fields

- const char * dn

  Distingushed name.

- struct bucket_list * bl

  bucket containing item to add

13.15.1 Detailed Description

LDAP Add structure.

Definition at line 76 of file openldap.c.

13.15.2 Field Documentation

13.15.2.1 struct bucket_list* ldap_add::bl

bucket containing item to add

Definition at line 80 of file openldap.c.

Referenced by ldap_addinit(), and ldap_doadd().

13.15.2.2 const char* ldap_add::dn

Distingushed name.

Definition at line 78 of file openldap.c.

Referenced by ldap_addinit(), and ldap_doadd().

The documentation for this struct was generated from the following file:

- src/openldap.c

13.16 ldap_attr Struct Reference

LDAP attirbute.

`#include <dtsapp.h>`

Data Fields

- const char * name

  Name of attribute.
- int count

  Value count.
- struct ldap_attrval ** vals

  Attribute value array.
- struct ldap_attr * next

  Next attribute.
- struct ldap_attr * prev

  Previous attribute.

13.16.1 Detailed Description

LDAP attirbute.

Definition at line 739 of file dtsapp.h.

13.16.2 Field Documentation

13.16.2.1 int ldap_attr::count

Value count.

Definition at line 743 of file dtsapp.h.

13.16.2.2 const char* ldap_attr::name

Name of attribute.

Definition at line 741 of file dtsapp.h.

13.16.2.3 struct ldap_attr* ldap_attr::next

Next attribute.

Definition at line 747 of file dtsapp.h.

Referenced by ldap_unref_attr().

13.16.2.4 struct ldap_attr* ldap_attr::prev

Previous attribute.

Definition at line 749 of file dtsapp.h.

13.16.2.5 struct ldap_attrval** ldap_attr::vals

Attribute value array.

Definition at line 745 of file dtsapp.h.

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.17 ldap_attrval Struct Reference

LDAP attribute value.

`#include <dtsapp.h>`

Data Fields

- int len

  Size of buffer.
- enum ldap_attrtype type

  Data type stored in buffer.
- char * buffer

  Value buffer.

13.17.1 Detailed Description

LDAP attribute value.

Definition at line 729 of file dtsapp.h.

13.17.2 Field Documentation

13.17.2.1 char* ldap_attrval::buffer

Value buffer.

Definition at line 735 of file dtsapp.h.

13.17.2.2 int ldap_attrval::len

Size of buffer.

Definition at line 731 of file dtsapp.h.

13.17.2.3 enum ldap_attrtype ldap_attrval::type

Data type stored in buffer.

Definition at line 733 of file dtsapp.h.

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.18 ldap_conn Struct Reference

LDAP connection.

Data Fields

- LDAP * ldap

  LDAP pointer.
- char * uri

  Address.
- int timelim

  Time limit.
- int limit

  Results limit.
- LDAPControl ** sctrlsp

  LDAP control.
- struct sasl_defaults * sasl

  SASL auth information.
- struct ldap_simple * simple

  LDAP Simple bind information.

13.18.1 Detailed Description

LDAP connection.

Definition at line 50 of file openldap.c.

13.18.2 Field Documentation

13.18.2.1 LDAP* ldap_conn::ldap

LDAP pointer.

Definition at line 52 of file openldap.c.

Referenced by ldap_connect(), ldap_doadd(), ldap_domodify(), ldap_saslbind(), and ldap_simplebind().

13.18.2.2 int ldap_conn::limit

Results limit.

Definition at line 58 of file openldap.c.

Referenced by ldap_connect().

13.18.2.3 struct sasl_defaults* ldap_conn::sasl

SASL auth information.

Definition at line 62 of file openldap.c.

Referenced by ldap_connect(), and ldap_saslbind().

13.18.2.4 LDAPControl** ldap_conn::sctrlsp

LDAP control.

Definition at line 60 of file openldap.c.

Referenced by ldap_connect(), ldap_doadd(), ldap_domodify(), ldap_saslbind(), and ldap_simplebind().

13.18.2.5 struct ldap_simple* ldap_conn::simple

LDAP Simple bind information.

Definition at line 64 of file openldap.c.

Referenced by ldap_simplebind().

13.18.2.6 int ldap_conn::timelim

Time limit.

Definition at line 56 of file openldap.c.

Referenced by ldap_connect().

13.18.2.7 char* ldap_conn::uri

Address.

Definition at line 54 of file openldap.c.

Referenced by ldap_connect().

The documentation for this struct was generated from the following file:

- src/openldap.c

13.19 ldap_entry Struct Reference

LDAP entry.

`#include <dtsapp.h>`

Data Fields

- const char * dn
  LDAP distiguished name.
- const char * dnufn
  LDAP user format distingushed name.
- int rdncnt
  RDN element count.
- struct ldap_rdn ** rdn
  RDN element array.
- struct ldap_attr * list
  Linked list of attributes.
- struct bucket_list * attrs
  Bucket list of attributes.
- struct ldap_attr * first_attr
  First attr (head of list).
- struct ldap_entry * next
  Next entry.
- struct ldap_entry * prev
  Previous entry.

13.19.1 Detailed Description

LDAP entry.

Definition at line 753 of file dtsapp.h.

13.19.2 Field Documentation

13.19.2.1 struct bucket_list* ldap_entry::attrs

Bucket list of attributes.

Definition at line 765 of file dtsapp.h.

Referenced by ldap_getattr(), and ldap_unref_attr().

13.19.2.2 const char* ldap_entry::dn

LDAP distiguished name.

Definition at line 755 of file dtsapp.h.

Referenced by ldap_simplerebind().

13.19.2.3 const char* ldap_entry::dnufn

LDAP user format distingushed name.

Definition at line 757 of file dtsapp.h.

13.19.2.4 struct ldap_attr* ldap_entry::first_attr

First attr (head of list).

Definition at line 767 of file dtsapp.h.

Referenced by ldap_unref_attr().

13.19.2.5 struct ldap_attr* ldap_entry::list

Linked list of attributes.

Definition at line 763 of file dtsapp.h.

13.19.2.6 struct ldap_entry* ldap_entry::next

Next entry.

Definition at line 769 of file dtsapp.h.

Referenced by ldap_unref_entry().

13.19.2.7 struct ldap_entry* ldap_entry::prev

Previous entry.

Definition at line 771 of file dtsapp.h.

13.19.2.8 struct ldap_rdn** ldap_entry::rdn

RDN element array.

Definition at line 761 of file dtsapp.h.

13.19.2.9 int ldap_entry::rdncnt

RDN element count.

Definition at line 759 of file dtsapp.h.

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.20 ldap_modify Struct Reference

LDAP Modify structure.

Data Fields

- const char * dn

  Distingushed name.
- struct bucket_list * bl [3]

  Bucket list containg modify / modify_add / delete requests.

13.20.1 Detailed Description

LDAP Modify structure.

Definition at line 68 of file openldap.c.

13.20.2 Field Documentation

13.20.2.1 struct bucket_list* ldap_modify::bl[3]

Bucket list containg modify / modify_add / delete requests.

Definition at line 72 of file openldap.c.

Referenced by ldap_domodify(), and ldap_modifyinit().

13.20.2.2 const char* ldap_modify::dn

Distingushed name.

Definition at line 70 of file openldap.c.

Referenced by ldap_domodify(), and ldap_modifyinit().

The documentation for this struct was generated from the following file:

- src/openldap.c

13.21 ldap_modreq Struct Reference

LDAP mod request.

Data Fields

- const char * attr
  Attribute modified.
- int cnt
  Count.
- struct ldap_modval * first
  Linked list head.
- struct ldap_modval * last
  Linked list tail.

13.21.1 Detailed Description

LDAP mod request.

Definition at line 92 of file openldap.c.

13.21.2 Field Documentation

13.21.2.1 const char* ldap_modreq::attr

Attribute modified.

Definition at line 94 of file openldap.c.

13.21.2.2 int ldap_modreq::cnt

Count.

Definition at line 96 of file openldap.c.

Referenced by ldap_domodify().

13.21.2.3 struct ldap_modval* ldap_modreq::first

Linked list head.

Definition at line 98 of file openldap.c.

13.21.2.4 struct ldap_modval* ldap_modreq::last

Linked list tail.

Definition at line 100 of file openldap.c.

The documentation for this struct was generated from the following file:

- src/openldap.c

13.22 ldap_modval Struct Reference

Linked list of mod values.

Data Fields

- const char * value
  Value.
- struct ldap_modval * next
  Next Value.

13.22.1 Detailed Description

Linked list of mod values.

Definition at line 84 of file openldap.c.

13.22.2 Field Documentation

13.22.2.1 struct ldap_modval* ldap_modval::next

Next Value.

Definition at line 88 of file openldap.c.

13.22.2.2 const char* ldap_modval::value

Value.

Definition at line 86 of file openldap.c.

The documentation for this struct was generated from the following file:

- src/openldap.c

13.23 ldap_rdn Struct Reference

LDAP Relative distingushed name linked list.

`#include <dtsapp.h>`

Data Fields

- const char * name
  RDN element name.
- const char * value
  RDN element value.
- struct ldap_rdn * next
  Next RDN element.
- struct ldap_rdn * prev
  Previous RDN element.

13.23.1 Detailed Description

LDAP Relative distingushed name linked list.

Definition at line 717 of file dtsapp.h.

13.23.2 Field Documentation

13.23.2.1 const char* ldap_rdn::name

RDN element name.

Definition at line 719 of file dtsapp.h.

13.23.2.2 struct ldap_rdn* ldap_rdn::next

Next RDN element.

Definition at line 723 of file dtsapp.h.

13.23.2.3 struct ldap_rdn* ldap_rdn::prev

Previous RDN element.

Definition at line 725 of file dtsapp.h.

13.23.2.4 const char* ldap_rdn::value

RDN element value.

Definition at line 721 of file dtsapp.h.

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.24 ldap_results Struct Reference

LDAP results.

`#include <dtsapp.h>`

Data Fields

- int count

  Number of entries.
- struct ldap_entry * first_entry

  Linked list of entries.
- struct bucket_list * entries

  Bucket list of entries.

13.24.1 Detailed Description

LDAP results.

Definition at line 775 of file dtsapp.h.

13.24.2 Field Documentation

13.24.2.1 int ldap_results::count

Number of entries.

Definition at line 777 of file dtsapp.h.

Referenced by ldap_simplerebind().

13.24.2.2 struct bucket_list* ldap_results::entries

Bucket list of entries.

Definition at line 781 of file dtsapp.h.

Referenced by ldap_getentry(), and ldap_unref_entry().

13.24.2.3 struct ldap_entry* ldap_results::first_entry

Linked list of entries.

Definition at line 779 of file dtsapp.h.

Referenced by ldap_simplerebind(), and ldap_unref_entry().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.25 ldap_simple Struct Reference

LDAP Simple bind.

Data Fields

- const char * dn

  Distingushed Name.
- struct berval * cred

  Credentials (password).

13.25.1 Detailed Description

LDAP Simple bind.

Definition at line 42 of file openldap.c.

13.25.2 Field Documentation

13.25.2.1 struct berval* ldap_simple::cred

Credentials (password).

Definition at line 46 of file openldap.c.

Referenced by ldap_simplebind().

13.25.2.2 const char* ldap_simple::dn

Distingushed Name.

Definition at line 44 of file openldap.c.

Referenced by ldap_simplebind().

The documentation for this struct was generated from the following file:

- src/openldap.c

13.26 natmap Struct Reference

RFC6296 Nat map.

Data Fields

- uint16_t mask

  The greater of internal or external subnet mask.
- uint16_t adjo

  Outbound adjustment.
- uint16_t adji

  Inbound adjustment.
- uint8_t ipre [16]

  Internal prefix.
- uint8_t epre [16]

  External prefix.

13.26.1 Detailed Description

RFC6296 Nat map.

Definition at line 33 of file rfc6296.c.

13.26.2 Field Documentation

13.26.2.1 uint16_t natmap::adji

Inbound adjustment.

Definition at line 39 of file rfc6296.c.

Referenced by rfc6296_map(), and rfc6296_map_add().

13.26.2.2 uint16_t natmap::adjo

Outbound adjustment.

Definition at line 37 of file rfc6296.c.

Referenced by rfc6296_map(), and rfc6296_map_add().

13.26.2.3 uint8_t natmap::epre[16]

External prefix.

Definition at line 43 of file rfc6296.c.

Referenced by rfc6296_map(), and rfc6296_map_add().

13.26.2.4 uint8_t natmap::ipre[16]

Internal prefix.

Definition at line 41 of file rfc6296.c.

Referenced by rfc6296_map(), and rfc6296_map_add().

13.26.2.5 uint16_t natmap::mask

The greater of internal or external subnet mask.

Definition at line 35 of file rfc6296.c.

Referenced by rfc6296_map(), and rfc6296_map_add().

The documentation for this struct was generated from the following file:

- src/rfc6296.c

13.27 nfq_queue Struct Reference

Data Fields

- struct nfq_struct * nfq
- struct nfq_q_handle * qh

- nfqueue_cb cb
- void * data
- uint16_t num

13.27.1 Detailed Description

Definition at line 54 of file nf_queue.c.

13.27.2 Field Documentation

13.27.2.1 nfqueue_cb nfq_queue::cb

Definition at line 57 of file nf_queue.c.

Referenced by nfqueue_attach().

13.27.2.2 void* nfq_queue::data

Definition at line 58 of file nf_queue.c.

Referenced by nfqueue_attach().

13.27.2.3 struct nfq_struct* nfq_queue::nfq

Definition at line 55 of file nf_queue.c.

Referenced by nfqueue_attach().

13.27.2.4 uint16_t nfq_queue::num

Definition at line 59 of file nf_queue.c.

13.27.2.5 struct nfq_q_handle* nfq_queue::qh

Definition at line 56 of file nf_queue.c.

Referenced by nfqueue_attach().

The documentation for this struct was generated from the following file:

- src/nf_queue.c

13.28 nfq_struct Struct Reference

Data Fields

- struct nfq_handle * h
- uint16_t pf
- int fd
- int flags

13.28.1 Detailed Description

Definition at line 47 of file nf_queue.c.

13.28.2 Field Documentation

13.28.2.1 int nfq_struct::fd

Definition at line 50 of file nf_queue.c.

13.28.2.2 int nfq_struct::flags

Definition at line 51 of file nf_queue.c.

13.28.2.3 struct nfq_handle* nfq_struct::h

Definition at line 48 of file nf_queue.c.

Referenced by nfqueue_attach().

13.28.2.4 uint16_t nfq_struct::pf

Definition at line 49 of file nf_queue.c.

The documentation for this struct was generated from the following file:

- src/nf_queue.c

13.29 pseudohdr Struct Reference

IPv4 header structur to cast a packet too.

Data Fields

- uint32_t saddr
  Source address.
- uint32_t daddr
  Destination address.
- uint8_t zero
  Zero byte.
- uint8_t proto
  protocol.
- uint16_t len
  Packet length.

13.29.1 Detailed Description

IPv4 header structur to cast a packet too.

Definition at line 84 of file iputil.c.

13.29.2 Field Documentation

13.29.2.1 uint32_t pseudohdr::daddr

Destination address.

Definition at line 88 of file iputil.c.

Referenced by ipv4tcpchecksum(), and ipv4udpchecksum().

13.29.2.2 uint16_t pseudohdr::len

Packet length.

Definition at line 94 of file iputil.c.

Referenced by ipv4tcpchecksum(), and ipv4udpchecksum().

13.29.2.3 uint8_t pseudohdr::proto

protocol.

Definition at line 92 of file iputil.c.

Referenced by ipv4tcpchecksum(), and ipv4udpchecksum().

13.29.2.4 uint32_t pseudohdr::saddr

Source address.

Definition at line 86 of file iputil.c.

Referenced by ipv4tcpchecksum(), and ipv4udpchecksum().

13.29.2.5 uint8_t pseudohdr::zero

Zero byte.

Definition at line 90 of file iputil.c.

Referenced by ipv4tcpchecksum(), and ipv4udpchecksum().

The documentation for this struct was generated from the following file:

- src/iputil.c

13.30 radius_connection Struct Reference

Radius connection.

Data Fields

- struct fwsocket * socket

  Reference to socket.
- unsigned char id

  Connection ID.
- struct radius_server * server

Reference to radius server.

- struct bucket_list * sessions

  Bucket list of sessions.

13.30.1 Detailed Description

Radius connection.

connect to the server one connection holds 256 sessions

Definition at line 89 of file radius.c.

13.30.2 Field Documentation

13.30.2.1 unsigned char radius_connection::id

Connection ID.

Definition at line 93 of file radius.c.

13.30.2.2 struct radius_server* radius_connection::server

Reference to radius server.

Definition at line 95 of file radius.c.

13.30.2.3 struct bucket_list* radius_connection::sessions

Bucket list of sessions.

Definition at line 97 of file radius.c.

13.30.2.4 struct fwsocket* radius_connection::socket

Reference to socket.

Definition at line 91 of file radius.c.

The documentation for this struct was generated from the following file:

- src/radius.c

13.31 radius_packet Struct Reference

Radius Packet.

Data Fields

- unsigned char code

  Radius packet code.

- unsigned char id

  Packet ID.

- unsigned short len

  Packet length.

- unsigned char token [RAD_AUTH_TOKEN_LEN]

  Authentification token.
- unsigned char attrs [RAD_AUTH_PACKET_LEN-RAD_AUTH_HDR_LEN]

  Radius Attributes.

13.31.1 Detailed Description

Radius Packet.

Definition at line 45 of file radius.c.

13.31.2 Field Documentation

13.31.2.1 unsigned char radius_packet::attrs[RAD_AUTH_PACKET_LEN-RAD_AUTH_HDR_LEN]

Radius Attributes.

Definition at line 56 of file radius.c.

Referenced by radius_attr_first(), and radius_attr_next().

13.31.2.2 unsigned char radius_packet::code

Radius packet code.

See Also

RADIUS_CODE.

Definition at line 48 of file radius.c.

Referenced by new_radpacket().

13.31.2.3 unsigned char radius_packet::id

Packet ID.

Definition at line 50 of file radius.c.

13.31.2.4 unsigned short radius_packet::len

Packet length.

Definition at line 52 of file radius.c.

Referenced by new_radpacket(), and radius_attr_next().

13.31.2.5 unsigned char radius_packet::token[RAD_AUTH_TOKEN_LEN]

Authentification token.

Definition at line 54 of file radius.c.

Referenced by new_radpacket().

The documentation for this struct was generated from the following file:

- src/radius.c

13.32 radius_server Struct Reference

Radius Server.

Data Fields

- const char * name
  Server name.
- const char * authport
  Server authport.
- const char * acctport
  Server accounting port.
- const char * secret
  Server secret.
- unsigned char id
  Server hash based on server count.
- int timeout
  Server timeout.
- struct timeval service
  Server out of service time.
- struct bucket_list * connex
  Bucket list of connextions.

13.32.1 Detailed Description

Radius Server.

define a server with host auth/acct port and secret create "connextions" on demand each with upto 256 sessions servers should not be removed without removing all and reloading

Definition at line 105 of file radius.c.

13.32.2 Field Documentation

13.32.2.1 const char* radius_server::acctport

Server accounting port.

Definition at line 111 of file radius.c.

Referenced by add_radserver().

13.32.2.2 const char* radius_server::authport

Server authport.

Definition at line 109 of file radius.c.

Referenced by add_radserver().

13.32.2.3 struct bucket_list* radius_server::connex

Bucket list of connextions.

Definition at line 121 of file radius.c.

13.32.2.4 unsigned char radius_server::id

Server hash based on server count.

Definition at line 115 of file radius.c.

Referenced by add_radserver().

13.32.2.5 const char* radius_server::name

Server name.

Definition at line 107 of file radius.c.

Referenced by add_radserver().

13.32.2.6 const char* radius_server::secret

Server secret.

Definition at line 113 of file radius.c.

Referenced by add_radserver().

13.32.2.7 struct timeval radius_server::service

Server out of service time.

Definition at line 119 of file radius.c.

Referenced by add_radserver().

13.32.2.8 int radius_server::timeout

Server timeout.

Definition at line 117 of file radius.c.

Referenced by add_radserver().

The documentation for this struct was generated from the following file:

- src/radius.c

13.33 radius_session Struct Reference

Radius session.

Data Fields

- unsigned short id

  Session id.
- unsigned char request [RAD_AUTH_TOKEN_LEN]

  Radius request auth token.
- void * cb_data

  Callback data passed to callback.
- radius_cb read_cb

Radius callback.

- unsigned int olen

  Original length of packet.

- struct radius_packet * packet

  Radius packet.

- struct timeval sent

  Time packet was sent.

- const char * passwd

  Password requires special handling.

- char retries

  Retries available.

- char minserver

  Minimum id of server to use.

13.33.1 Detailed Description

Radius session.

A radius session is based on a ID packet for session stored till a response the request token is also stored

Definition at line 63 of file radius.c.

13.33.2 Field Documentation

13.33.2.1 void* radius_session::cb_data

Callback data passed to callback.

Definition at line 69 of file radius.c.

13.33.2.2 unsigned short radius_session::id

Session id.

Definition at line 65 of file radius.c.

13.33.2.3 char radius_session::minserver

Minimum id of server to use.

Definition at line 83 of file radius.c.

13.33.2.4 unsigned int radius_session::olen

Original length of packet.

Definition at line 73 of file radius.c.

13.33.2.5 struct radius_packet* radius_session::packet

Radius packet.

Definition at line 75 of file radius.c.

13.33.2.6 const char* radius_session::passwd

Password requires special handling.

Definition at line 79 of file radius.c.

13.33.2.7 radius_cb radius_session::read_cb

Radius callback.

Definition at line 71 of file radius.c.

13.33.2.8 unsigned char radius_session::request[RAD_AUTH_TOKEN_LEN]

Radius request auth token.

Definition at line 67 of file radius.c.

13.33.2.9 char radius_session::retries

Retries available.

Definition at line 81 of file radius.c.

13.33.2.10 struct timeval radius_session::sent

Time packet was sent.

Definition at line 77 of file radius.c.

The documentation for this struct was generated from the following file:

- src/radius.c

13.34 ref_obj Struct Reference

Internal structure of all referenced objects.

Data Fields

- uint32_t magic

  Memory integrity check used to prevent non refeferenced objects been handled as referenced objects.
- uint32_t cnt

  Reference count the oject will be freed when the reference count reaches 0.
- size_t size

  The size allocated to this object.
- pthread_mutex_t lock

  this is a pointer to the lock it may be changed to be the lock
- objdestroy destroy

  Function to call to clean up the data before its freed.
- void * data

  Pointer to the data referenced.

13.34.1 Detailed Description

Internal structure of all referenced objects.

Definition at line 38 of file refobj.c.

13.34.2 Field Documentation

13.34.2.1 uint32_t ref_obj::cnt

Reference count the oject will be freed when the reference count reaches 0.

Definition at line 45 of file refobj.c.

Referenced by objalloc(), objcnt(), objref(), and objunref().

13.34.2.2 void* ref_obj::data

Pointer to the data referenced.

Definition at line 54 of file refobj.c.

Referenced by addtobucket(), bucket_list_find_key(), next_bucket_loop(), objalloc(), objcnt(), objlock(), objref(), objsize(), objtrylock(), objunlock(), objunref(), remove_bucket_item(), and remove_bucket_loop().

13.34.2.3 objdestroy ref_obj::destroy

Function to call to clean up the data before its freed.

Definition at line 52 of file refobj.c.

Referenced by objalloc(), and objunref().

13.34.2.4 pthread_mutex_t ref_obj::lock

this is a pointer to the lock it may be changed to be the lock

Definition at line 50 of file refobj.c.

Referenced by objalloc(), objcnt(), objlock(), objref(), objsize(), objtrylock(), objunlock(), and objunref().

13.34.2.5 uint32_t ref_obj::magic

Memory integrity check used to prevent non refeferenced objects been handled as referenced objects.

See Also

REFOBJ_MAGIC

Definition at line 42 of file refobj.c.

Referenced by objalloc(), objcnt(), objlock(), objref(), objsize(), objtrylock(), objunlock(), and objunref().

13.34.2.6 size_t ref_obj::size

The size allocated to this object.

Warning

this may be removed in future.

Definition at line 48 of file refobj.c.

Referenced by objalloc(), objsize(), and objunref().

The documentation for this struct was generated from the following file:

- src/refobj.c

13.35 sasl_defaults Struct Reference

SASL Paramaters used in authentification.

Data Fields

- const char * mech

  SASL Mechanisim.
- const char * realm

  SASL Realm.
- const char * authcid

  Auth ID.
- const char * passwd

  Password.
- const char * authzid

  Proxy auth ID.

13.35.1 Detailed Description

SASL Paramaters used in authentification.

Definition at line 28 of file openldap.c.

13.35.2 Field Documentation

13.35.2.1 const char* sasl_defaults::authcid

Auth ID.

Definition at line 34 of file openldap.c.

Referenced by ldap_saslbind().

13.35.2.2 const char* sasl_defaults::authzid

Proxy auth ID.

Definition at line 38 of file openldap.c.

Referenced by ldap_saslbind().

13.35.2.3 const char* sasl_defaults::mech

SASL Mechanisim.

Definition at line 30 of file openldap.c.

Referenced by ldap_saslbind().

13.35.2.4 const char* sasl_defaults::passwd

Password.

Definition at line 36 of file openldap.c.

Referenced by ldap_saslbind().

13.35.2.5 const char* sasl_defaults::realm

SASL Realm.

Definition at line 32 of file openldap.c.

Referenced by ldap_saslbind().

The documentation for this struct was generated from the following file:

- src/openldap.c

13.36 socket_handler Struct Reference

Socket handling thread data.

Data Fields

- struct fwsocket * sock

  Socket this thread manages.

- void * data

  Reference to data passed in callbacks.

- socketrecv client

  Callback called when the socket is ready to read.

- threadcleanup cleanup

  Callback to call when the thread closes to allow additional cleanup.

- socketrecv connect

  If a client connects to a bound port this callback is called on connect.

13.36.1 Detailed Description

Socket handling thread data.

Definition at line 51 of file socket.c.

13.36.2 Field Documentation

13.36.2.1 threadcleanup socket_handler::cleanup

Callback to call when the thread closes to allow additional cleanup.

Definition at line 60 of file socket.c.

13.36.2.2 socketrecv socket_handler::client

Callback called when the socket is ready to read.

Definition at line 57 of file socket.c.

13.36.2.3 socketrecv socket_handler::connect

If a client connects to a bound port this callback is called on connect.

Definition at line 63 of file socket.c.

13.36.2.4 void* socket_handler::data

Reference to data passed in callbacks.

Definition at line 55 of file socket.c.

13.36.2.5 struct fwsocket* socket_handler::sock

Socket this thread manages.

Definition at line 53 of file socket.c.

The documentation for this struct was generated from the following file:

- src/socket.c

13.37 sockstruct Union Reference

Socket union describing all address types.

`#include <dtsapp.h>`

Data Fields

- struct sockaddr sa
  Base socket addr structure.
- struct sockaddr_un un
  Unix sockets.
- struct sockaddr_in sa4
  IPv4 socket addr structure.
- struct sockaddr_in6 sa6
  IPv6 socket addr structure.
- struct sockaddr_storage ss
  Sockaddr storage is a "magic" struct been able to hold IPv4 or IPv6.

13.37.1 Detailed Description

Socket union describing all address types.

Examples:

socket.c.

Definition at line 80 of file dtsapp.h.

13.37.2 Field Documentation

13.37.2.1 struct sockaddr sockstruct::sa

Base socket addr structure.

Definition at line 82 of file dtsapp.h.

Referenced by accept_socket(), dtls_listenssl(), inet_ntop(), mcast_socket(), socketread_d(), and socketwrite_d().

13.37.2.2 struct sockaddr_in sockstruct::sa4

IPv4 socket addr structure.

Definition at line 88 of file dtsapp.h.

Referenced by inet_ntop(), and sockaddr2ip().

13.37.2.3 struct sockaddr_in6 sockstruct::sa6

IPv6 socket addr structure.

Definition at line 90 of file dtsapp.h.

Referenced by inet_ntop(), and sockaddr2ip().

13.37.2.4 struct sockaddr_storage sockstruct::ss

Sockaddr storage is a "magic" struct been able to hold IPv4 or IPv6.

Definition at line 92 of file dtsapp.h.

Referenced by inet_ntop(), sockaddr2ip(), and socketwrite_d().

13.37.2.5 struct sockaddr_un sockstruct::un

Unix sockets.

Definition at line 85 of file dtsapp.h.

Referenced by socketwrite_d(), and unixsocket_client().

The documentation for this union was generated from the following file:

- src/include/dtsapp.h

13.38 sslData Struct Reference

SSL data structure for enabling encryption on sockets.

Data Fields

- SSL_CTX * ctx

  OpenSSL context.
- SSL * ssl

  OpenSSL ssl.
- BIO * bio

  OpenSSL BIO.
- int flags

  SSL flags.
- const SSL_METHOD * meth

  SSL method.
- struct ssldata * parent

  Parent structure.

13.38.1 Detailed Description

SSL data structure for enabling encryption on sockets.

Definition at line 66 of file sslutil.c.

13.38.2 Field Documentation

13.38.2.1 BIO* ssldata::bio

OpenSSL BIO.

Definition at line 72 of file sslutil.c.

13.38.2.2 SSL_CTX* ssldata::ctx

OpenSSL context.

Definition at line 68 of file sslutil.c.

Referenced by dtlsv1_init(), and dtsl_serveropts().

13.38.2.3 int ssldata::flags

SSL flags.

See Also

SSLFLAGS

Definition at line 75 of file sslutil.c.

Referenced by dtls_listenssl(), dtsl_serveropts(), and startsslclient().

13.38.2.4 const SSL_METHOD* ssldata::meth

SSL method.

Definition at line 77 of file sslutil.c.

13.38.2.5 struct ssldata* ssldata::parent

Parent structure.

Definition at line 79 of file sslutil.c.

13.38.2.6 SSL* ssldata::ssl

OpenSSL ssl.

Definition at line 70 of file sslutil.c.

Referenced by dtls_listenssl(), dtlshandltimeout(), dtlstimeout(), dtlsv1_init(), dtsl_serveropts(), socketread_d(), socketwrite_d(), ssl_shutdown(), and sslv3_init().

The documentation for this struct was generated from the following file:

- src/sslutil.c

13.39 thread_pvt Struct Reference

thread struct used to create threads data needs to be first element

Data Fields

- void * data
  Reference to data held on thread creation.
- pthread_t thr
  Thread information.
- threadcleanup cleanup
  Thread cleanup callback.
- threadfunc func
  Thread function.
- threadsighandler sighandler
  Thread signal handler.
- enum threadopt flags
  thread options

13.39.1 Detailed Description

thread struct used to create threads data needs to be first element

Definition at line 58 of file thread.c.

13.39.2 Field Documentation

13.39.2.1 threadcleanup thread_pvt::cleanup

Thread cleanup callback.

See Also

threadcleanup

Definition at line 65 of file thread.c.

Referenced by framework_mkthread().

13.39.2.2 void* thread_pvt::data

Reference to data held on thread creation.

Definition at line 60 of file thread.c.

Referenced by framework_mkthread().

13.39.2.3 enum threadopt thread_pvt::flags

thread options

See Also

threadopt_flags

Definition at line 74 of file thread.c.

Referenced by framework_mkthread().

13.39.2.4 threadfunc thread_pvt::func

Thread function.

See Also

threadfunc

Definition at line 68 of file thread.c.

Referenced by framework_mkthread().

13.39.2.5 threadsighandler thread_pvt::sighandler

Thread signal handler.

See Also

threadsighandler

Definition at line 71 of file thread.c.

Referenced by framework_mkthread().

13.39.2.6 pthread_t thread_pvt::thr

Thread information.

Definition at line 62 of file thread.c.

Referenced by framework_mkthread(), framework_threadok(), jointhreads(), and stopthreads().

The documentation for this struct was generated from the following file:

- src/thread.c

13.40 threadcontainer Struct Reference

Global threads data.

Data Fields

- struct bucket_list * list
  Hashed bucket list of threads.
- struct thread_pvt * manager
  Manager thread.

13.40.1 Detailed Description

Global threads data.

Definition at line 78 of file thread.c.

13.40.2 Field Documentation

13.40.2.1 struct bucket_list* threadcontainer::list

Hashed bucket list of threads.

Definition at line 80 of file thread.c.

Referenced by framework_mkthread(), and startthreads().

13.40.2.2 struct thread_pvt* threadcontainer::manager

Manager thread.

Definition at line 82 of file thread.c.

Referenced by framework_mkthread(), jointhreads(), startthreads(), and stopthreads().

The documentation for this struct was generated from the following file:

- src/thread.c

13.41 unixclient_sockthread Struct Reference

Unix socket client data structure.

Data Fields

- struct fwsocket * sock
  Socket reference.
- socketrecv client
  Client read callback.
- const char * endpoint
  Client endpoint tmp for SOCK_DGRAM.
- void * data
  Data reference passed to callback.

13.41.1 Detailed Description

Unix socket client data structure.

Definition at line 63 of file unixsock.c.

13.41.2 Field Documentation

13.41.2.1 socketrecv unixclient_sockthread::client

Client read callback.

See Also

socketrecv

Definition at line 68 of file unixsock.c.

13.41.2.2 void* unixclient_sockthread::data

Data reference passed to callback.

Definition at line 72 of file unixsock.c.

13.41.2.3 const char* unixclient_sockthread::endpoint

Client endpoint tmp for SOCK_DGRAM.

Definition at line 70 of file unixsock.c.

13.41.2.4 struct fwsocket* unixclient_sockthread::sock

Socket reference.

Definition at line 65 of file unixsock.c.

The documentation for this struct was generated from the following file:

- src/unixsock.c

13.42 unixserv_sockthread Struct Reference

Unix socket server data structure.

Data Fields

- struct fwsocket * sock
  Socket reference.
- char sockpath [UNIX_PATH_MAX+1]
  Socket path.
- int mask
  Socket umask.
- int protocol
  Socket protocol.
- socketrecv read
  Thread to begin on client connect.
- void * data
  Data reference passed to callback.

13.42.1 Detailed Description

Unix socket server data structure.

Definition at line 46 of file unixsock.c.

13.42.2 Field Documentation

13.42.2.1 void* unixserv_sockthread::data

Data reference passed to callback.

Definition at line 59 of file unixsock.c.

Referenced by unixsocket_server().

13.42.2.2 int unixserv_sockthread::mask

Socket umask.

Definition at line 52 of file unixsock.c.

Referenced by unixsocket_server().

13.42.2.3 int unixserv_sockthread::protocol

Socket protocol.

Definition at line 54 of file unixsock.c.

Referenced by unixsocket_server().

13.42.2.4 socketrecv unixserv_sockthread::read

Thread to begin on client connect.

See Also

threadfunc

Definition at line 57 of file unixsock.c.

Referenced by unixsocket_server().

13.42.2.5 struct fwsocket* unixserv_sockthread::sock

Socket reference.

Definition at line 48 of file unixsock.c.

Referenced by unixsocket_server().

13.42.2.6 char unixserv_sockthread::sockpath[UNIX_PATH_MAX+1]

Socket path.

Definition at line 50 of file unixsock.c.

Referenced by unixsocket_server().

The documentation for this struct was generated from the following file:

- src/unixsock.c

13.43 xml_attr Struct Reference

XML attribute name value pair.

`#include <dtsapp.h>`

Data Fields

- const char * name
  Name of attribute.
- const char * value
  Value of attribute.

13.43.1 Detailed Description

XML attribute name value pair.

Definition at line 639 of file dtsapp.h.

13.43.2 Field Documentation

13.43.2.1 const char* xml_attr::name

Name of attribute.

Definition at line 641 of file dtsapp.h.

13.43.2.2 const char* xml_attr::value

Value of attribute.

Definition at line 643 of file dtsapp.h.

Referenced by xml_getattr().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.44 xml_node Struct Reference

Reference to a XML Node.

`#include <dtsapp.h>`

Data Fields

- const char * name
  Name of the node.
- const char * value
  Value of the node.

- const char * key

  Attribute key for searching and indexing.
- struct bucket_list * attrs

  Bucket list of attributes.
- void * nodeptr

  Internal libxml2 node pointer.

13.44.1 Detailed Description

Reference to a XML Node.

Definition at line 648 of file dtsapp.h.

13.44.2 Field Documentation

13.44.2.1 struct bucket_list* xml_node::attrs

Bucket list of attributes.

Definition at line 656 of file dtsapp.h.

Referenced by xml_getattr().

13.44.2.2 const char* xml_node::key

Attribute key for searching and indexing.

Definition at line 654 of file dtsapp.h.

13.44.2.3 const char* xml_node::name

Name of the node.

Definition at line 650 of file dtsapp.h.

Referenced by xml_createpath().

13.44.2.4 void* xml_node::nodeptr

Internal libxml2 node pointer.

Definition at line 658 of file dtsapp.h.

Referenced by xml_appendnode(), xml_delete(), xml_modify(), xml_setattr(), and xml_unlink().

13.44.2.5 const char* xml_node::value

Value of the node.

Definition at line 652 of file dtsapp.h.

Referenced by xml_modify().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

13.45 xml_node_iter Struct Reference

Iterator to traverse nodes in a xpath.

Data Fields

- struct xml_search * xsearch

  Reference to search returned from xml_search()
- int curpos

  current position.
- int cnt

  number of nodes in search path.

13.45.1 Detailed Description

Iterator to traverse nodes in a xpath.

Definition at line 22 of file libxml2.c.

13.45.2 Field Documentation

13.45.2.1 int xml_node_iter::cnt

number of nodes in search path.

Definition at line 28 of file libxml2.c.

Referenced by xml_getfirstnode(), and xml_getnextnode().

13.45.2.2 int xml_node_iter::curpos

current position.

Definition at line 26 of file libxml2.c.

Referenced by xml_getfirstnode(), and xml_getnextnode().

13.45.2.3 struct xml_search* xml_node_iter::xsearch

Reference to search returned from xml_search()

Definition at line 24 of file libxml2.c.

Referenced by xml_getfirstnode(), and xml_getnextnode().

The documentation for this struct was generated from the following file:

- src/libxml2.c

13.46 xml_search Struct Reference

XML xpath search result.

Data Fields

- struct xml_doc * xmldoc

  Reference to XML document.
- xmlXPathObjectPtr xpathObj

  Xpath object.
- struct bucket_list * nodes

  Bucket list of all nodes.

13.46.1 Detailed Description

XML xpath search result.

See Also

xml_search()

Definition at line 33 of file libxml2.c.

13.46.2 Field Documentation

13.46.2.1 struct bucket_list* xml_search::nodes

Bucket list of all nodes.

Definition at line 39 of file libxml2.c.

Referenced by xml_getnode(), xml_getnodes(), and xml_xpath().

13.46.2.2 struct xml_doc* xml_search::xmldoc

Reference to XML document.

Definition at line 35 of file libxml2.c.

Referenced by xml_xpath().

13.46.2.3 xmlXPathObjectPtr xml_search::xpathObj

Xpath object.

Definition at line 37 of file libxml2.c.

Referenced by xml_nodecount(), and xml_xpath().

The documentation for this struct was generated from the following file:

- src/libxml2.c

13.47 xslt_doc Struct Reference

XSLT Document.

Data Fields

- xsltStylesheetPtr doc

  Pointer to the document.
- struct bucket_list * params

  Bucket list of paramaters to apply to the document.

13.47.1 Detailed Description

XSLT Document.

Definition at line 21 of file libxslt.c.

13.47.2 Field Documentation

13.47.2.1 xsltStylesheetPtr xslt_doc::doc

Pointer to the document.

Definition at line 23 of file libxslt.c.

Referenced by xslt_apply(), xslt_apply_buffer(), and xslt_open().

13.47.2.2 struct bucket_list* xslt_doc::params

Bucket list of paramaters to apply to the document.

Definition at line 25 of file libxslt.c.

Referenced by xslt_addparam(), xslt_clearparam(), and xslt_open().

The documentation for this struct was generated from the following file:

- src/libxslt.c

13.48 xslt_param Struct Reference

XSLT Parameter name/value pair.

Data Fields

- const char * name

  Name of paramater.
- const char * value

  value of paramater.

13.48.1 Detailed Description

XSLT Parameter name/value pair.

Definition at line 29 of file libxslt.c.

13.48.2 Field Documentation

13.48.2.1 const char* xslt_param::name

Name of paramater.

Definition at line 31 of file libxslt.c.

Referenced by xslt_addparam().

13.48.2.2 const char* xslt_param::value

value of paramater.

Definition at line 33 of file libxslt.c.

Referenced by xslt_addparam().

The documentation for this struct was generated from the following file:

- src/libxslt.c

13.49 zobj Struct Reference

Zlib buffer used for compression and decompression.

`#include <dtsapp.h>`

Data Fields

- uint8_t * buff

  Buffer with compressed/uncompressed data.
- uint16_t olen

  Original size of data.
- uint16_t zlen

  Compressed size of data.

13.49.1 Detailed Description

Zlib buffer used for compression and decompression.

Definition at line 164 of file dtsapp.h.

13.49.2 Field Documentation

13.49.2.1 uint8_t* zobj::buff

Buffer with compressed/uncompressed data.

Definition at line 166 of file dtsapp.h.

Referenced by zcompress(), and zuncompress().

13.49.2.2 uint16_t zobj::olen

Original size of data.

Definition at line 168 of file dtsapp.h.

Referenced by zcompress(), and zuncompress().

13.49.2.3 uint16_t zobj::zlen

Compressed size of data.

Definition at line 170 of file dtsapp.h.

Referenced by zcompress(), and zuncompress().

The documentation for this struct was generated from the following file:

- src/include/dtsapp.h

Chapter 14

File Documentation

14.1 doxygen/dox/buckets.dox File Reference

14.2 doxygen/dox/examples.dox File Reference

14.3 doxygen/dox/index.dox File Reference

14.4 doxygen/dox/main.dox File Reference

14.5 doxygen/dox/modules.dox File Reference

14.6 doxygen/dox/refobj.dox File Reference

14.7 doxygen/dox/sockets.dox File Reference

14.8 doxygen/dox/sockex.dox File Reference

14.9 doxygen/dox/thread.dox File Reference

14.10 doxygen/examples/socket.c File Reference

Echo server using 1 server and 2 clients.

```
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#include <openssl/ssl.h>
#include <dtsapp.h>
```

Functions

- void accept_func (struct fwsocket *sock, void *data)

  This function does nothing and is here for completeness.

- void server_func (struct fwsocket ∗sock, void ∗data)

  Server thread data is available.
- void client_func (struct fwsocket ∗sock, void ∗data)

  client thread data is available.
- void socktest (const char ∗ipaddr, int tcp, int ssl)

  Bassed on the options create server and clients.
- void unixsocktest (const char ∗socket, int protocol)

  Same test as for socktest() but for unix domain sockets.
- FRAMEWORK_MAIN ("Socket Client/Server Echo (TCP/TLS/UDP/DTLS)","Gregory Hinton Nietsky","gregory@distrotech.co.za","http://www.distrotech.co.za", 2013,"/var/run/sockettest", FRAMEWORK_FLAG_DAEMONLOCK, NULL)

  Initialise the application under the library replacing main()

14.10.1 Detailed Description

Echo server using 1 server and 2 clients. Simple implementation of a echo server shoeing the network socket interface it creates 1 server and 2 client threads the server echos back what is sent. the sockets support ipv4 and ipv6 and can be UDP or TCP with or without TLS/SSL support.

On application start using FRAMEWORK_MAIN a licence banner is displayed no flags are set as i wish to daemonize after checking the command line arguments.

There is a run/lock file created failure to lock this file prevents execution.

Once the sockets are created and threads started i sleep the main thread for 5 seconds before exiting the system will make sure all threads stop before leaving.

As you can see the progam initiliztion and flow has been greatly simplified by having these tasks managed.

Definition in file socket.c.

14.10.2 Function Documentation

14.10.2.1 void accept_func (struct fwsocket ∗ *sock*, void ∗ *data*)

This function does nothing and is here for completeness.

When a new connection is recieved this function will be executed to allow processing of the connection.

Parameters

sock	Reference to new socket
data	Reference to data suppled on thread start

Examples:

socket.c.

Definition at line 36 of file socket.c.

Referenced by socktest().

```
00036                                                   {
00037 }
```

14.10.2.2 void client_func (struct fwsocket ∗ *sock*, void ∗ *data*)

client thread data is available.

There is no need to worry about UDP support in client trhead callbacks and use of socketread / socketwrite is all that is required.

Parameters

sock	Reference to socket data is available on.
data	Reference to data held by thread.

Examples:

socket.c.

Definition at line 66 of file socket.c.

References fwsocket::sock, socketread(), and socketwrite().

Referenced by socktest(), and unixsocktest().

```
00066                                                   {
00067     char buff[128];
00068
00069     if (socketread(sock, &buff, 128) > 0) {
00070         socketwrite(sock, &buff, strlen(buff) + 1);
00071         printf("[C] %s %i\n", buff, sock->sock);
00072     }
00073 }
```

14.10.2.3 FRAMEWORK_MAIN ("Socket Client/Server Echo (TCP/TLS/UDP/DTLS)" , "Gregory Hinton Nietsky" , "gregory@distrotech.co.za" , "http://www.distrotech.co.za" , 2013 , "/var/run/sockettest" , FRAMEWORK_FLAG_DAEMONLOCK , NULL)

Initialise the application under the library replacing main()

See Also

FRAMEWORK_MAIN()
framework_mkcore()
framework_init()

[main] [main]

Examples:

socket.c.

Definition at line 167 of file socket.c.

References daemonize(), socktest(), and unixsocktest().

```
00168             ://www.distrotech.co.za", 2013, "/var/run/sockettest", FRAMEWORK_FLAG_DAEMONLOCK, NULL) {
00169
00170     if (argc < 3) {
00171 #ifndef __WIN32
00172         printf("Requires arguments %s [tcp|tls|udp|dtls|unix_d|unix_s] [ipaddr|socket]\n", argv[0]);
00173 #else
00174         printf("Requires arguments %s [tcp|tls|udp|dtls] ipaddr\n", argv[0]);
00175 #endif
00176         return (-1);
00177     }
00178
00179     daemonize();
00182     if (!strcmp(argv[1], "udp")) {
00183         socktest(argv[2], 0, 0);
00184     } else if (!strcmp(argv[1], "dtls")) {
00185         socktest(argv[2], 0, 1);
00186     } else if (!strcmp(argv[1], "tcp")) {
00187         socktest(argv[2], 1, 0);
00188     } else if (!strcmp(argv[1], "tls")) {
00189         socktest(argv[2], 1, 1);
00190 #ifndef __WIN32
00191     } else if (!strcmp(argv[1], "unix_d")) {
00192         unixsocktest(argv[2], SOCK_DGRAM);
```

```
00193     } else if (!strcmp(argv[1], "unix_s")) {
00194         unixsocktest(argv[2], SOCK_STREAM);
00195 #endif
00196     } else {
00197         printf("Invalid Option\n");
00198     }
00199 }
```

14.10.2.4 void server_func (struct fwsocket * *sock*, void * *data*)

Server thread data is available.

This function executes when the server socket has data to read the socket will need to be read from using socketread[_d] socketread_d is a wrapper arround recvfrom and socketwerite_d is a wrapper arround sendto this is important when dealing with un encrypted UDP sessions where the socket needs sendto addresss to send data too.

Parameters

sock	Reference to socket data is available on.
data	Reference to data held by thread.

Examples:

socket.c.

Definition at line 48 of file socket.c.

References fwsocket::sock, socketread_d(), and socketwrite_d().

Referenced by socktest(), and unixsocktest().

```
00048                                                   {
00049     char buff[128];
00050     union sockstruct addr;
00051
00052     if (socketread_d(sock, &buff, 128, &addr) > 0) {
00053         socketwrite_d(sock, &buff, strlen(buff) + 1, &addr);
00054         printf("[S] %s %i\n", buff, sock->sock);
00055         sleep(1);
00056     }
00057 }
```

14.10.2.5 void socktest (const char * *ipaddr*, int *tcp*, int *ssl*)

Bassed on the options create server and clients.

- If SSL / TLS was requested create SSL/TLS sessions to use.
- Bind to server and connect the clients.
- Start threads.
- Send data to server.
- Sleep

Parameters

ipaddr	As supplied on the command line
tcp	Set to non zero if using TCP.
ssl	Set to non zero if TLS/SSL is required.

Examples:

socket.c.

Definition at line 86 of file socket.c.

References accept_func(), client_func(), close_socket(), dtlsv1_init(), server_func(), socketclient(), socketserver(), socketwrite(), sslv3_init(), tcpbind(), tcpconnect(), udpbind(), and udpconnect().

Referenced by FRAMEWORK_MAIN().

```
00086                                                   {
00087     struct fwsocket *serv, *client, *client2;
00088     void *ssl_c = NULL, *ssl_s = NULL, *ssl_c2 = NULL;
00089     char *buff = "client 1";
00090     char *buff2 = "client 2";
00091     int cnt;
00092
00093     if (ssl && tcp) {
00094         ssl_s = sslv3_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
      SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
00095         ssl_c = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
00096         ssl_c2 = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
00097     } else if (ssl) {
00098         ssl_s = dtlsv1_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
       SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
00099         ssl_c = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
       SSL_VERIFY_NONE);
00100         ssl_c2 = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem"
      , SSL_VERIFY_NONE);
00101     }
00102
00103     if (tcp) {
00104         serv = tcpbind(ipaddr, "1111", ssl_s, 10);
00105         client = tcpconnect(ipaddr, "1111", ssl_c);
00106         client2 = tcpconnect(ipaddr, "1111", ssl_c2);
00107     } else {
00108         serv = udpbind(ipaddr, "1111", ssl_s);
00109         client = udpconnect(ipaddr, "1111", ssl_c);
00110         client2 = udpconnect(ipaddr, "1111", ssl_c2);
00111     }
00112
00113     if (serv && client && client2) {
00114         socketserver(serv, server_func, accept_func, NULL, NULL);
00115         socketclient(client, NULL, client_func, NULL);
00116         socketclient(client2, NULL, client_func, NULL);
00117
00118         socketwrite(client, buff, strlen(buff)+1);
00119         socketwrite(client2, buff2, strlen(buff2)+1);
00120
00121         sleep(5);
00122     } else {
00123         printf("ERROR\n");
00124     }
00125
00126     close_socket(client);
00127     close_socket(client2);
00128     close_socket(serv);
00129 }
```

14.10.2.6 void unixsocktest (const char * *socket*, int *protocol*)

Same test as for socktest() but for unix domain sockets.

Unix domain sockets are "file" sockets and function in similar way to network sockets there scope is local to the machine so are often used for inter process control and networkless services. Instead of a IP address a file name is specified that is created by the server.

Parameters

socket	File name to create server on and connect too.
protocol	Theis is either SOCK_STREAM or SOCK_DGRAM and are similar to TCP/UDP respectivly.

Examples:

socket.c.

Definition at line 139 of file socket.c.

References client_func(), close_socket(), server_func(), socketwrite_d(), unixsocket_client(), and unixsocket_server().

Referenced by FRAMEWORK_MAIN().

```
00139                                                   {
00140     char *buff = "client 1";
00141     char *buff2 = "client 2";
00142     struct fwsocket *client, *client2, *server;
00143 
00144     server = unixsocket_server(socket, protocol, S_IXUSR | S_IWGRP | S_IRGRP | S_IXGRP |
      S_IWOTH | S_IROTH | S_IXOTH, server_func, NULL);
00145     sleep(1); /*wait for socket*/
00146     client = unixsocket_client(socket, protocol, client_func, NULL);
00147     client2 = unixsocket_client(socket, protocol, client_func, NULL);
00148 
00149     socketwrite_d(client, buff, strlen(buff)+1, NULL);
00150     socketwrite_d(client2, buff2, strlen(buff2)+1, NULL);
00151 
00152     sleep(5);
00153 
00154     close_socket(client);
00155     close_socket(client2);
00156     close_socket(server);
00157 }
```

14.11 socket.c

```
00001 #ifdef __WIN32
00002 #include <winsock2.h>
00003 #include <stdint.h>
00004 #else
00005 #include <fcntl.h>
00006 #endif
00007 
00008 #include <string.h>
00009 #include <stdio.h>
00010 
00011 #include <openssl/ssl.h>
00012 #include <dtsapp.h>
00013 
00036 void accept_func(struct fwsocket *sock, void *data) {
00037 }
00038 
00048 void server_func(struct fwsocket *sock, void *data) {
00049     char buff[128];
00050     union sockstruct addr;
00051 
00052     if (socketread_d(sock, &buff, 128, &addr) > 0) {
00053         socketwrite_d(sock, &buff, strlen(buff) + 1, &addr);
00054         printf("[S] %s %i\n", buff, sock->sock);
00055         sleep(1);
00056     }
00057 }
00058 
00066 void client_func(struct fwsocket *sock, void *data) {
00067     char buff[128];
00068 
00069     if (socketread(sock, &buff, 128) > 0) {
00070         socketwrite(sock, &buff, strlen(buff) + 1);
00071         printf("[C] %s %i\n", buff, sock->sock);
00072     }
00073 }
00074 
00086 void socktest(const char *ipaddr, int tcp, int ssl) {
00087     struct fwsocket *serv, *client, *client2;
00088     void *ssl_c = NULL, *ssl_s = NULL, *ssl_c2 = NULL;
```

```
00089     char *buff = "client 1";
00090     char *buff2 = "client 2";
00091     int cnt;
00092 
00093     if (ssl && tcp) {
00094         ssl_s = sslv3_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
      SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
00095         ssl_c = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
00096         ssl_c2 = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
00097     } else if (ssl) {
00098         ssl_s = dtlsv1_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
       SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
00099         ssl_c = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
       SSL_VERIFY_NONE);
00100         ssl_c2 = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem"
      , SSL_VERIFY_NONE);
00101     }
00102 
00103     if (tcp) {
00104         serv = tcpbind(ipaddr, "1111", ssl_s, 10);
00105         client = tcpconnect(ipaddr, "1111", ssl_c);
00106         client2 = tcpconnect(ipaddr, "1111", ssl_c2);
00107     } else {
00108         serv = udpbind(ipaddr, "1111", ssl_s);
00109         client = udpconnect(ipaddr, "1111", ssl_c);
00110         client2 = udpconnect(ipaddr, "1111", ssl_c2);
00111     }
00112 
00113     if (serv && client && client2) {
00114         socketserver(serv, server_func, accept_func, NULL, NULL);
00115         socketclient(client, NULL, client_func, NULL);
00116         socketclient(client2, NULL, client_func, NULL);
00117 
00118         socketwrite(client, buff, strlen(buff)+1);
00119         socketwrite(client2, buff2, strlen(buff2)+1);
00120 
00121         sleep(5);
00122     } else {
00123         printf("ERROR\n");
00124     }
00125 
00126     close_socket(client);
00127     close_socket(client2);
00128     close_socket(serv);
00129 }
00130 
00138 #ifndef __WIN32
00139 void unixsocktest(const char *socket, int protocol) {
00140     char *buff = "client 1";
00141     char *buff2 = "client 2";
00142     struct fwsocket *client, *client2, *server;
00143 
00144     server = unixsocket_server(socket, protocol, S_IXUSR | S_IWGRP | S_IRGRP | S_IXGRP |
      S_IWOTH | S_IROTH | S_IXOTH, server_func, NULL);
00145     sleep(1); /*wait for socket*/
00146     client = unixsocket_client(socket, protocol, client_func, NULL);
00147     client2 = unixsocket_client(socket, protocol, client_func, NULL);
00148 
00149     socketwrite_d(client, buff, strlen(buff)+1, NULL);
00150     socketwrite_d(client2, buff2, strlen(buff2)+1, NULL);
00151 
00152     sleep(5);
00153 
00154     close_socket(client);
00155     close_socket(client2);
00156     close_socket(server);
00157 }
00158 #endif
00159 
00167 FRAMEWORK_MAIN("Socket Client/Server Echo (TCP/TLS/UDP/DTLS)", "Gregory Hinton Nietsky", "
      gregory@distrotech.co.za",
00168         "http://www.distrotech.co.za", 2013, "/var/run/sockettest",
      FRAMEWORK_FLAG_DAEMONLOCK, NULL) {
00169 
00170     if (argc < 3) {
00171 #ifndef __WIN32
00172         printf("Requires arguments %s [tcp|tls|udp|dtls|unix_d|unix_s] [ipaddr|socket]\n", argv[0]);
00173 #else
00174         printf("Requires arguments %s [tcp|tls|udp|dtls] ipaddr\n", argv[0]);
00175 #endif
00176         return (-1);
00177     }
00178 
00179     daemonize();
00182     if (!strcmp(argv[1], "udp")) {
```

```
00183         socktest(argv[2], 0, 0);
00184     } else if (!strcmp(argv[1], "dtls")) {
00185         socktest(argv[2], 0, 1);
00186     } else if (!strcmp(argv[1], "tcp")) {
00187         socktest(argv[2], 1, 0);
00188     } else if (!strcmp(argv[1], "tls")) {
00189         socktest(argv[2], 1, 1);
00190 #ifndef __WIN32
00191     } else if (!strcmp(argv[1], "unix_d")) {
00192         unixsocktest(argv[2], SOCK_DGRAM);
00193     } else if (!strcmp(argv[1], "unix_s")) {
00194         unixsocktest(argv[2], SOCK_STREAM);
00195 #endif
00196     } else {
00197         printf("Invalid Option\n");
00198     }
00199 }
```

14.12 src/socket.c File Reference

Allocate and initialise a socket for use as a client or server.

```
#include <netdb.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include "include/dtsapp.h"
#include "include/private.h"
```

Data Structures

- struct socket_handler

  Socket handling thread data.

Functions

- void close_socket (struct fwsocket *sock)

  Mark the socket for closure and release the reference.
- struct fwsocket * make_socket (int family, int type, int proto, void *ssl)

  Allocate a socket structure and return reference.
- struct fwsocket * accept_socket (struct fwsocket *sock)

  Create and return a socket structure from accept()
- struct fwsocket * sockconnect (int family, int stype, int proto, const char *ipaddr, const char *port, void *ssl)

  Generic client socket.
- struct fwsocket * udpconnect (const char *ipaddr, const char *port, void *ssl)

  UDP Socket client.
- struct fwsocket * tcpconnect (const char *ipaddr, const char *port, void *ssl)

  TCP Socket client.
- struct fwsocket * sockbind (int family, int stype, int proto, const char *ipaddr, const char *port, void *ssl, int backlog)

  Generic server socket.
- struct fwsocket * udpbind (const char *ipaddr, const char *port, void *ssl)

UDP server socket.

- struct fwsocket * tcpbind (const char *ipaddr, const char *port, void *ssl, int backlog)

  Generic server socket.
- void socketserver (struct fwsocket *sock, socketrecv read, socketrecv acceptfunc, threadcleanup cleanup, void *data)

  Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.
- void socketclient (struct fwsocket *sock, void *data, socketrecv read, threadcleanup cleanup)

  Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.
- const char * sockaddr2ip (union sockstruct *addr, char *buff, int blen)

  Return the ip address of a sockstruct addr.
- struct fwsocket * mcast_socket (const char *iface, int family, const char *mcastip, const char *port, int flags)

  Create a multicast socket.

14.12.1 Detailed Description

Allocate and initialise a socket for use as a client or server. This is part of the socket interface to upport encrypted sockets a ssldata refernece will be created and passed on socket initialization.

See Also

SSL socket support

Definition in file socket.c.

14.13 socket.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00030 #ifndef __WIN32__
00031 #include <netdb.h>
00032 #endif
00033 #include <unistd.h>
00034 #include <stdint.h>
00035 #include <string.h>
00036 #include <errno.h>
00037 #include <stdio.h>
00038 #include <fcntl.h>
00039 #ifdef __WIN32__
00040 #include <winsock2.h>
00041 #include <ws2tcpip.h>
00042 #else
00043 #include <arpa/inet.h>
00044 #endif
00045 #include <stdlib.h>
00046
00047 #include "include/dtsapp.h"
00048 #include "include/private.h"
00049
00051 struct socket_handler {
00053     struct fwsocket *sock;
00055     void *data;
00057     socketrecv    client;
00060     threadcleanup  cleanup;
```

```
00063     socketrecv    connect;
00064 };
00065
00066 static int32_t hash_socket(const void *data, int key) {
00067     int ret;
00068     const struct fwsocket *sock = data;
00069     const int *hashkey = (key) ? data : &sock->sock;
00070
00071     ret = *hashkey;
00072
00073     return (ret);
00074 }
00075
00079 extern void close_socket(struct fwsocket *sock) {
00080     if (sock) {
00081         setflag(sock, SOCK_FLAG_CLOSE);
00082         objunref(sock);
00083     }
00084 }
00085
00086 static void clean_fwsocket(void *data) {
00087     struct fwsocket *sock = data;
00088
00089     if (sock->ssl) {
00090         objunref(sock->ssl);
00091     }
00092
00093     /*im closing remove from parent list*/
00094     if (sock->parent) {
00095         if (sock->parent->children) {
00096             remove_bucket_item(sock->parent->children, sock);
00097         }
00098         objunref(sock->parent);
00099     }
00100
00101     /*looks like the server is shut down*/
00102     if (sock->children) {
00103         objunref(sock->children);
00104     }
00105
00106     if (sock->sock >= 0) {
00107         close(sock->sock);
00108     }
00109 }
00110
00120 extern struct fwsocket *make_socket(int family, int type, int
      proto, void *ssl) {
00121     struct fwsocket *si;
00122
00123     if (!(si = objalloc(sizeof(*si),clean_fwsocket))) {
00124         return NULL;
00125     }
00126
00127     if ((si->sock = socket(family, type, proto)) < 0) {
00128         objunref(si);
00129         return NULL;
00130     };
00131
00132     if (ssl) {
00133         si->ssl = ssl;
00134     }
00135     si->type = type;
00136     si->proto = proto;
00137
00138     return (si);
00139 }
00140
00144 extern struct fwsocket *accept_socket(struct fwsocket *sock) {
00145     struct fwsocket *si;
00146     socklen_t salen = sizeof(si->addr);
00147
00148     if (!(si = objalloc(sizeof(*si),clean_fwsocket))) {
00149         return NULL;
00150     }
00151
00152     objlock(sock);
00153     if ((si->sock = accept(sock->sock, &si->addr.sa, &salen)) < 0) {
00154         objunlock(sock);
00155         objunref(si);
00156         return NULL;
00157     }
00158
00159     si->type = sock->type;
00160     si->proto = sock->proto;
00161
00162     if (sock->ssl) {
00163         tlsaccept(si, sock->ssl);
```

```
00164     }
00165     objunlock(sock);
00166
00167     return (si);
00168 }
00169
00170 static struct fwsocket *_opensocket(int family, int stype, int proto, const char *ipaddr,
      const char *port, void *ssl, int ctype, int backlog) {
00171     struct  addrinfo hint, *result, *rp;
00172     struct fwsocket *sock = NULL;
00173     socklen_t salen = sizeof(union sockstruct);
00174 #ifndef __WIN32__
00175     int on = 1;
00176 #endif
00177
00178     memset(&hint, 0, sizeof(hint));
00179     hint.ai_family = family;
00180     hint.ai_socktype = stype;
00181     hint.ai_protocol = proto;
00182
00183     if (getaddrinfo(ipaddr, port, &hint, &result) || !result) {
00184         return (NULL);
00185     }
00186
00187     for(rp = result; rp; rp = result->ai_next) {
00188         if (!(sock = make_socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol, ssl))) {
00189             continue;
00190         }
00191         if (ctype) {
00192 #ifndef __WIN32__
00193             setsockopt(sock->sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
00194 #ifdef SO_REUSEPORT
00195             setsockopt(sock->sock, SOL_SOCKET, SO_REUSEPORT, &on, sizeof(on));
00196 #endif
00197 #else
00198 /*      ioctlsocket(sock->sock, SO_REUSEADDR, (unsigned long*)&on);*/
00199 #endif
00200         }
00201         if ((!ctype && !connect(sock->sock, rp->ai_addr, rp->ai_addrlen)) ||
00202                 (ctype && !bind(sock->sock, rp->ai_addr, rp->ai_addrlen))) {
00203             break;
00204         }
00205         objunref(sock);
00206         sock = NULL;
00207     }
00208
00209
00210     if (!sock || !rp) {
00211         if (sock) {
00212             objunref(sock);
00213         }
00214         freeaddrinfo(result);
00215
00216         return (NULL);
00217     }
00218
00219
00220     if (ctype) {
00221         sock->flags |= SOCK_FLAG_BIND;
00222         memcpy(&sock->addr.ss, rp->ai_addr, sizeof(sock->addr.ss));
00223         switch(sock->type) {
00224             case SOCK_STREAM:
00225             case SOCK_SEQPACKET:
00226                 listen(sock->sock, backlog);
00227                 /* no break */
00228             default:
00229                 break;
00230         }
00231     } else {
00232         getsockname(sock->sock, &sock->addr.sa, &salen);
00233     }
00234
00235     freeaddrinfo(result);
00236     return (sock);
00237 }
00238
00250 extern struct fwsocket *sockconnect(int family, int stype, int proto, const char *ipaddr
      , const char *port, void *ssl) {
00251     return(_opensocket(family, stype, proto, ipaddr, port, ssl, 0, 0));
00252 }
00253
00262 extern struct fwsocket *udpconnect(const char *ipaddr, const char *port, void *ssl) {
00263     return (_opensocket(PF_UNSPEC, SOCK_DGRAM, IPPROTO_UDP, ipaddr, port, ssl, 0, 0));
00264 }
00265
00274 extern struct fwsocket *tcpconnect(const char *ipaddr, const char *port, void *ssl) {
00275     return (_opensocket(PF_UNSPEC, SOCK_STREAM, IPPROTO_TCP, ipaddr, port, ssl, 0, 0));
```

```
00276 }
00277
00290 extern struct fwsocket *sockbind(int family, int stype, int proto, const char *ipaddr,
      const char *port, void *ssl, int backlog) {
00291     return(_opensocket(family, stype, proto, ipaddr, port, ssl, 1, backlog));
00292 }
00293
00302 extern struct fwsocket *udpbind(const char *ipaddr, const char *port, void *ssl) {
00303     return (_opensocket(PF_UNSPEC, SOCK_DGRAM, IPPROTO_UDP, ipaddr, port, ssl, 1, 0));
00304 }
00305
00315 extern struct fwsocket *tcpbind(const char *ipaddr, const char *port, void *ssl, int backlog
      ) {
00316     return (_opensocket(PF_UNSPEC, SOCK_STREAM, IPPROTO_TCP, ipaddr, port, ssl, 1, backlog));
00317 }
00318
00319 static void _socket_handler_clean(void *data) {
00320     struct socket_handler *fwsel = data;
00321
00322     /*call cleanup and remove refs to data*/
00323     if (fwsel->cleanup) {
00324         fwsel->cleanup(fwsel->data);
00325     }
00326     if (fwsel->data) {
00327         objunref(fwsel->data);
00328     }
00329 }
00330
00331 static void *_socket_handler(void *data) {
00332     struct socket_handler *sockh = data;
00333     struct fwsocket *sock = sockh->sock;
00334     struct fwsocket *newsock;
00335     struct  timeval tv;
00336     fd_set  rd_set, act_set;
00337     int selfd, sockfd, type, flags;
00338     struct bucket_loop *bloop;
00339 #ifdef __WIN32
00340     int errcode;
00341 #endif
00342     objlock(sock);
00343     FD_ZERO(&rd_set);
00344     sockfd = sock->sock;
00345     type = sock->type;
00346     if ((sock->flags & SOCK_FLAG_BIND) && (sock->ssl || !(sock->
      type == SOCK_DGRAM))) {
00347         flags = (SOCK_FLAG_BIND & sock->flags);
00348     } else {
00349         flags = 0;
00350     }
00351     FD_SET(sockfd, &rd_set);
00352     objunlock(sock);
00353
00354     while (framework_threadok() && !testflag(sock,
      SOCK_FLAG_CLOSE)) {
00355         act_set = rd_set;
00356         tv.tv_sec = 0;
00357         tv.tv_usec = 20000;
00358
00359         selfd = select(sockfd + 1, &act_set, NULL, NULL, &tv);
00360
00361         /*returned due to interupt continue or timed out*/
00362 #ifndef __WIN32
00363         if ((selfd < 0 && errno == EINTR) || (!selfd)) {
00364 #else
00365         errcode = WSAGetLastError();
00366         if (((selfd == SOCKET_ERROR) && (errcode == WSAEINTR)) || (!selfd)) {
00367 #endif
00368             if ((type == SOCK_DGRAM) && (flags & SOCK_FLAG_BIND)) {
00369                 dtlshandltimeout(sock);
00370             }
00371             continue;
00372         } else if (selfd < 0) {
00373             break;
00374         }
00375
00376         if (FD_ISSET(sockfd, &act_set)) {
00377             if (flags & SOCK_FLAG_BIND) {
00378                 switch (type) {
00379                     case SOCK_STREAM:
00380                     case SOCK_SEQPACKET:
00381                         newsock = accept_socket(sock);
00382                         break;
00383                     case SOCK_DGRAM:
00384                         newsock = dtls_listenssl(sock);
00385                         break;
00386                     default:
00387                         newsock = NULL;
```

```
00388                       break;
00389                   }
00390                   if (newsock) {
00391                       objref(sock);
00392                       newsock->parent = sock;
00393                       addtobucket(sock->children, newsock);
00394                       socketclient(newsock, sockh->data, sockh->
      client, NULL);
00395                       if (sockh->connect) {
00396                           sockh->connect(newsock, sockh->data);
00397                       }
00398                       objunref(newsock); /*pass ref to thread*/
00399                   }
00400               } else {
00401                   sockh->client(sockh->sock, sockh->data);
00402               }
00403           }
00404       }
00405
00406       if (sock->ssl) {
00407           ssl_shutdown(sock->ssl, sock->sock);
00408       }
00409
00410       /*close children*/
00411       if (sock->children) {
00412           bloop = init_bucket_loop(sock->children);
00413           while(bloop && (newsock = next_bucket_loop(bloop))) {
00414               remove_bucket_loop(bloop);
00415               objlock(newsock);
00416               if (newsock->parent) {
00417                   objunref(newsock->parent);
00418                   newsock->parent = NULL;
00419               }
00420               objunlock(newsock);
00421               close_socket(newsock); /*remove ref*/
00422           }
00423           objunref(bloop);
00424       }
00425
00426       objunref(sock);
00427
00428       return NULL;
00429   }
00430
00431   static void _start_socket_handler(struct fwsocket *sock, socketrecv read,
00432                                     socketrecv acceptfunc, threadcleanup cleanup, void
       *data) {
00433       struct socket_handler *sockh;
00434
00435       if (!sock || !read || !(sockh = objalloc(sizeof(*sockh), NULL))) {
00436           return;
00437       }
00438
00439       sockh->sock = sock;
00440       sockh->client = read;
00441       sockh->cleanup = cleanup;
00442       sockh->connect = acceptfunc;
00443       sockh->data = data;
00444
00445       /* grab ref for data and pass sockh*/
00446       objref(data);
00447       objref(sock);
00448       framework_mkthread(_socket_handler, _socket_handler_clean, NULL, sockh, 0);
00449       objunref(sockh);
00450   }
00451
00463   extern void socketserver(struct fwsocket *sock, socketrecv read,
00464                            socketrecv acceptfunc, threadcleanup cleanup, void *data) {
00465
00466       objlock(sock);
00467       if (sock->flags & SOCK_FLAG_BIND) {
00468           if (sock->ssl || !(sock->type == SOCK_DGRAM)) {
00469               sock->children = create_bucketlist(6, hash_socket);
00470           }
00471           if (sock->ssl && (sock->type == SOCK_DGRAM)) {
00472               objunlock(sock);
00473               dtsl_serveropts(sock);
00474           } else {
00475               objunlock(sock);
00476           }
00477       } else {
00478           objunlock(sock);
00479       }
00480       _start_socket_handler(sock, read, acceptfunc, cleanup, data);
00481   }
00482
00493   extern void socketclient(struct fwsocket *sock, void *data,
```

```
    socketrecv read, threadcleanup cleanup) {
00494     startsslclient(sock);
00495
00496     _start_socket_handler(sock, read, NULL, cleanup, data);
00497 }
00498
00504 const char *sockaddr2ip(union sockstruct *addr, char *buff, int blen) {
00505     if (!buff) {
00506         return NULL;
00507     }
00508
00509     switch (addr->ss.ss_family) {
00510         case PF_INET:
00511             inet_ntop(PF_INET, &addr->sa4.sin_addr, buff, blen);
00512             break;
00513         case PF_INET6:
00514             inet_ntop(PF_INET6, &addr->sa6.sin6_addr, buff, blen);
00515             break;
00516     }
00517     return buff;
00518 }
00519
00536 struct fwsocket *mcast_socket(const char *iface, int family, const char *mcastip, const
     char *port, int flags) {
00537     struct fwsocket *fws;
00538     struct  addrinfo hint, *result, *rp;
00539     struct in_addr *srcif;
00540     const char *srcip;
00541     int ifidx;
00542     int on = 1;
00543     int off = 0;
00544     int ttl = 50;
00545     socklen_t slen = sizeof(union sockstruct);
00546 #ifdef __WIN32
00547     struct ifinfo *ifinf;
00548 #endif
00549
00550         memset(&hint, 0, sizeof(hint));
00551     hint.ai_family = PF_UNSPEC;
00552     hint.ai_socktype = SOCK_DGRAM;
00553     hint.ai_protocol = IPPROTO_UDP;
00554
00555 #ifndef __WIN32
00556     if (!(srcip = get_ifipaddr(iface, family))) {
00557             return NULL;
00558     }
00559
00560         if (getaddrinfo(srcip, port, &hint, &result) || !result) {
00561         free((void*)srcip);
00562             return NULL;
00563         }
00564     free((void*)srcip);
00565 #else
00566     if (!(ifinf = get_ifinfo(iface))) {
00567         return NULL;
00568     }
00569     ifidx = ifinf->idx;
00570
00571     srcip = (family == AF_INET) ? ifinf->ipv4addr : ifinf->ipv6addr;
00572         if (!srcip || (getaddrinfo(srcip, port, &hint, &result) || !result)) {
00573         objunref(ifinf);
00574             return NULL;
00575         }
00576     objunref(ifinf);
00577 #endif
00578
00579     for(rp = result; rp; rp = result->ai_next) {
00580         if (!(fws = make_socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol, NULL))) {
00581             continue;
00582         }
00583         break;
00584     }
00585
00586     if (!rp || !fws) {
00587         freeaddrinfo(result);
00588         return NULL;
00589     }
00590
00591     if(setsockopt(fws->sock, SOL_SOCKET, SO_REUSEADDR, (char*)&on, sizeof(on))) {
00592         objunref(fws);
00593         freeaddrinfo(result);
00594         return NULL;
00595     }
00596
00597     if (rp->ai_family == PF_INET) {
00598         struct in_addr mcastip4;
00599         struct ip_mreq mg;
```

```
00600         struct sockaddr_in *src_ip;
00601
00602         src_ip = (struct sockaddr_in*)rp->ai_addr;
00603
00604         if (setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_TTL, (char*)&ttl, sizeof(ttl))) {
00605             objunref(fws);
00606             freeaddrinfo(result);
00607             return NULL;
00608         }
00609
00610         if (flags && setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_LOOP, (char*)&off, sizeof(off))) {
00611             freeaddrinfo(result);
00612             objunref(fws);
00613             return NULL;
00614         }
00615
00616         if (mcastip) {
00617             inet_lookup(PF_INET, mcastip, &mcastip4, sizeof(mcastip4));
00618         } else {
00619             seedrand();
00620             mcast4_ip(&mcastip4);
00621         }
00622
00623         mg.imr_multiaddr = mcastip4;
00624         mg.imr_interface.s_addr = src_ip->sin_addr.s_addr;
00625         if (setsockopt(fws->sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char*)&mg, sizeof(mg))) {
00626             objunref(fws);
00627             freeaddrinfo(rp);
00628             return NULL;
00629         }
00630
00631         memset(&srcif, 0, sizeof(srcif));
00632         srcif = &src_ip->sin_addr;
00633         if(setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_IF, (char*)srcif, sizeof(*srcif))) {
00634             freeaddrinfo(rp);
00635             objunref(fws);
00636             return NULL;
00637         }
00638         src_ip->sin_addr.s_addr = mcastip4.s_addr;
00639     } else if (rp->ai_family == PF_INET6) {
00640         struct in6_addr mcastip6;
00641         struct ipv6_mreq mg;
00642         struct sockaddr_in6 *src_ip;
00643
00644 #ifndef __WIN32
00645         ifidx = get_iface_index(iface);
00646 #endif
00647         src_ip = (struct sockaddr_in6*)rp->ai_addr;
00648
00649         if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_HOPS, (char*)&ttl, sizeof(ttl))) {
00650             objunref(fws);
00651             freeaddrinfo(result);
00652             return NULL;
00653         }
00654
00655         if (flags && setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_LOOP, (char*)&off, sizeof(off))
    ) {
00656             freeaddrinfo(result);
00657             objunref(fws);
00658             return NULL;
00659         }
00660
00661         if (mcastip) {
00662             inet_lookup(PF_INET6, mcastip, &mcastip6, sizeof(mcastip6));
00663         } else {
00664             seedrand();
00665             mcast6_ip(&mcastip6);
00666         }
00667
00668         mg.ipv6mr_multiaddr = mcastip6;
00669         mg.ipv6mr_interface = ifidx;
00670         if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_JOIN_GROUP, (char*)&mg, sizeof(mg))) {
00671             objunref(fws);
00672             freeaddrinfo(rp);
00673             return NULL;
00674         }
00675
00676         if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_IF, (char*)&ifidx, sizeof(ifidx))) {
00677             objref(fws);
00678             freeaddrinfo(rp);
00679             return NULL;
00680         }
00681
00682         src_ip->sin6_addr = mcastip6;
00683     }
00684
00685     if (bind(fws->sock, (struct sockaddr*)rp->ai_addr, sizeof(struct sockaddr_storage))) {
```

```
00686         freeaddrinfo(result);
00687         objunref(fws);
00688         return NULL;
00689     }
00690
00691     getsockname(fws->sock, &fws->addr.sa, &slen);
00692     freeaddrinfo(result);
00693     fws->flags |= SOCK_FLAG_MCAST;
00694
00695     return fws;
00696 }
00697
00698
```

14.14 src/config.c File Reference

INI style config file interface.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include "include/dtsapp.h"
```

Data Structures

- struct config_category

  Configuration file category.
- struct config_file

  Config file.

Functions

- void unrefconfigfiles (void)

  Empty out and unreference config files.
- int process_config (const char *configname, const char *configfile)

  Process a configfile into buckets.
- struct bucket_list * get_config_file (const char *configname)

  Returns the catergories bucket for a config file.
- struct bucket_list * get_config_category (const char *configname, const char *category)

  Return a single category.
- struct bucket_list * get_category_next (struct bucket_loop *cloop, char *name, int len)

  Iterate through categories returning the entries bucket.
- struct bucket_loop * get_category_loop (const char *configname)

  Return a bucket loop to allow iterating over categories.
- void config_entry_callback (struct bucket_list *entries, config_entrycb entry_cb)

  Callback Wraper that iterates through all items calling a callback for each item.
- void config_cat_callback (struct bucket_list *categories, config_catcb cat_cb)

  Callback wrapper that iterates through categories calling a callback on each category.
- void config_file_callback (config_filecb file_cb)

  Callback wrapper to iterate over all configfiles calling a callback on each file.
- struct config_entry * get_config_entry (struct bucket_list *categories, const char *item)

  Find the entry in a config file.

14.14.1 Detailed Description

INI style config file interface.

Definition in file config.c.

14.15 config.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00025 #include <stdio.h>
00026 #include <stdlib.h>
00027 #include <stdint.h>
00028 #include <string.h>
00029 #include "include/dtsapp.h"
00030
00032 struct config_category {
00034     const char *name;
00036     struct bucket_list *entries;
00037 };
00038
00040 struct config_file {
00042     const char *filename;
00044     const char *filepath;
00046     struct bucket_list *cat;
00047 };
00048
00049 static struct bucket_list *configfiles = NULL;
00050
00051 static int32_t hash_files(const void *data, int key) {
00052     int ret;
00053     const struct config_file *file = data;
00054     const char *hashkey = (key) ? data : file->filename;
00055
00056     ret = jenhash(hashkey, strlen(hashkey), 0);
00057
00058     return(ret);
00059 }
00060
00061 static int32_t hash_cats(const void *data, int key) {
00062     int ret;
00063     const struct config_category *cat = data;
00064     const char *hashkey = (key) ? data : cat->name;
00065
00066     ret = jenhash(hashkey, strlen(hashkey), 0);
00067
00068     return(ret);
00069 }
00070
00071 static void initconfigfiles(void) {
00072     if (!configfiles) {
00073         configfiles = create_bucketlist(4, hash_files);
00074     }
00075 }
00076
00078 extern void unrefconfigfiles(void) {
00079     if (configfiles) {
00080         objunref(configfiles);
00081     }
00082 }
00083
00084 static void free_config_entry(void *data) {
00085     struct config_entry *entry = data;
00086
00087     if (entry->item) {
00088         free((void *)entry->item);
```

```
00089     }
00090     if (entry->value) {
00091         free((void *)entry->value);
00092     }
00093 }
00094
00095 static void add_conf_entry(struct config_category *category, const char *
      item, const char *value) {
00096     struct config_entry *newentry;
00097
00098     if (!category || !category->entries || !(newentry = objalloc(sizeof(*newentry),
      free_config_entry))) {
00099         return;
00100     }
00101
00102     ALLOC_CONST(newentry->item, item);
00103     ALLOC_CONST(newentry->value, value);
00104
00105     addtobucket(category->entries, newentry);
00106     objunref(newentry);
00107 }
00108
00109 static void free_config_category(void *data) {
00110     struct config_category *cat = data;
00111
00112     if (cat->name) {
00113         free((void *)cat->name);
00114     }
00115     if (cat->entries) {
00116         objunref(cat->entries);
00117     }
00118 }
00119
00120 static struct config_category *create_conf_category(const char *
      name) {
00121     struct config_category *newcat;
00122
00123     if (!(newcat = objalloc(sizeof(*newcat), free_config_category))) {
00124         return (NULL);
00125     }
00126
00127     ALLOC_CONST(newcat->name, name);
00128     newcat->entries = create_bucketlist(5, hash_cats);
00129
00130     return (newcat);
00131 }
00132
00133 static void free_config_file(void *data) {
00134     struct config_file *file = data;
00135
00136     if (file->filename) {
00137         free((void *)file->filename);
00138     }
00139     if (file->filepath) {
00140         free((void *)file->filepath);
00141     }
00142     if (file->cat) {
00143         objunref(file->cat);
00144     }
00145 }
00146
00147 static struct config_file *create_conf_file(const char *filename, const char *
      filepath) {
00148     struct config_file *newfile;
00149
00150     if (!(newfile = objalloc(sizeof(*newfile), free_config_file))) {
00151         return (NULL);
00152     }
00153
00154     ALLOC_CONST(newfile->filename, filename);
00155     ALLOC_CONST(newfile->filepath, filepath);
00156     newfile->cat = create_bucketlist(4, hash_files);
00157
00158     return (newfile);
00159 }
00160
00161 static char *filterconf(const char *str, int minlen) {
00162     char *tmp, *token;
00163
00164     /*trim leading and trailing white space*/
00165     tmp = trim(str);
00166
00167     /*remove everything after the last # ignore if # is first*/
00168     if ((token = strrchr(tmp, '#'))) {
00169         if (token == tmp) {
00170             return NULL;
00171         }
```

```
00172         token[0] = '\0';
00173     }
00174
00175     /*first char is #*/
00176     if ((token = strchr(tmp, '#')) && (token == tmp)) {
00177         return NULL;
00178     }
00179
00180     /*remove ; as first char*/
00181     if ((token = strchr(tmp, ';')) && (token == tmp)) {
00182         return NULL;
00183     }
00184
00185     /*too short*/
00186     if (strlen(tmp) < minlen) {
00187         return NULL;
00188     }
00189
00190     return (tmp);
00191 }
00192
00197 extern int process_config(const char *configname, const char *configfile) {
00198     struct config_file *file;
00199     struct config_category *category = NULL;
00200     FILE *config;
00201     char line[256];
00202     char item[128];
00203     char value[128];
00204     char *tmp = (char *)&line;
00205     char *token;
00206
00207     if (!configfiles) {
00208         initconfigfiles();
00209     }
00210
00211     file = create_conf_file(configname, configfile);
00212     addtobucket(configfiles, file);
00213
00214     if (!(config = fopen(file->filepath, "r"))) {
00215         return (-1);
00216     }
00217
00218     while(fgets(line, sizeof(line) - 1, config)) {
00219         if (!(tmp = filterconf(line, 3))) {
00220             continue;
00221         }
00222
00223         /*this is a new category*/
00224         if ((token = strchr(tmp, '[')) && (token == tmp)) {
00225             tmp++;
00226             token = strrchr(tmp, ']');
00227             token[0] = '\0';
00228             tmp = trim(tmp);
00229             if (!strlenzero(tmp)) {
00230                 if (category) {
00231                     objunref(category);
00232                 }
00233                 category = create_conf_category(tmp);
00234                 addtobucket(file->cat, category);
00235             }
00236             continue;
00237         }
00238
00239         if (sscanf(tmp, "%[^=] %*[=] %[^\n]", (char *)&item, (char *)&value) != 2) {
00240             continue;
00241         }
00242
00243         if (!category) {
00244             category = create_conf_category("default");
00245             addtobucket(file->cat, category);
00246         }
00247
00248         add_conf_entry(category, trim(item), trim(value));
00249     }
00250     fclose(config);
00251     if (category) {
00252         objunref(category);
00253     }
00254     if (file) {
00255         objunref(file);
00256     }
00257     return (0);
00258 }
00259
00263 extern struct bucket_list *get_config_file(const char *configname) {
00264     struct config_file *file;
00265
```

```
00266     if ((file = bucket_list_find_key(configfiles, configname))) {
00267         if (file->cat) {
00268             if (!objref(file->cat)) {
00269                 objunref(file);
00270                 return (NULL);
00271             }
00272             objunref(file);
00273             return (file->cat);
00274         }
00275         objunref(file);
00276     }
00277     return (NULL);
00278 }
00279
00286 extern struct bucket_list *get_config_category(const char *configname, const
      char *category) {
00287     struct bucket_list *file;
00288     struct config_category *cat;
00289
00290     file = get_config_file(configname);
00291     if (category) {
00292         cat = bucket_list_find_key(file, category);
00293     } else {
00294         cat = bucket_list_find_key(file, "default");
00295     }
00296
00297     objunref(file);
00298     if (cat) {
00299         if (!objref(cat->entries)) {
00300             objunref(cat);
00301             return (NULL);
00302         }
00303         objunref(cat);
00304         return (cat->entries);
00305     } else {
00306         return (NULL);
00307     }
00308 }
00309
00317 extern struct bucket_list *get_category_next(struct
      bucket_loop *cloop, char *name, int len) {
00318     struct config_category *category;
00319
00320     if (cloop && (category = next_bucket_loop(cloop))) {
00321         if (category->entries) {
00322             if (!objref(category->entries)) {
00323                 objunref(category);
00324                 return (NULL);
00325             }
00326             if (!strlenzero(name)) {
00327                 strncpy(name, category->name, len);
00328             }
00329             objunref(category);
00330             return (category->entries);
00331         } else {
00332             objunref(category);
00333         }
00334     }
00335     return (NULL);
00336 }
00337
00341 extern struct bucket_loop *get_category_loop(const char *configname) {
00342     struct bucket_loop *cloop;
00343     struct bucket_list *file;
00344
00345     file = get_config_file(configname);
00346     cloop = init_bucket_loop(file);
00347     objunref(file);
00348     return (cloop);
00349 }
00350
00351 static void entry_callback(void *data, void *entry_cb) {
00352     struct config_entry *entry = data;
00353     config_entrycb *cb_entry = entry_cb, callback;
00354
00355     callback = *cb_entry;
00356
00357     callback(entry->item, entry->value);
00358 }
00359
00365 extern void config_entry_callback(struct bucket_list *entries,
      config_entrycb entry_cb) {
00366     bucketlist_callback(entries, entry_callback, &entry_cb);
00367 }
00368
00369 static void category_callback(void *data, void *category_cb) {
00370     struct config_category *category = data;
```

```
00371     config_catcb *cb_catptr = category_cb, cb_cat;
00372
00373     cb_cat = *cb_catptr;
00374
00375     cb_cat(category->entries, category->name);
00376 }
00377
00383 extern void config_cat_callback(struct bucket_list *categories,
      config_catcb cat_cb) {
00384     bucketlist_callback(categories, category_callback, &cat_cb);
00385 }
00386
00387 static void file_callback(void *data, void *file_cb) {
00388     struct config_file *file = data;
00389     config_filecb *cb_fileptr = file_cb, cb_file;
00390
00391     cb_file = *cb_fileptr;
00392
00393     cb_file(file->cat, file->filename, file->filepath);
00394 }
00395
00400 extern void config_file_callback(config_filecb file_cb) {
00401     bucketlist_callback(configfiles, file_callback, &file_cb);
00402 }
00403
00408 extern struct config_entry *get_config_entry(struct
      bucket_list *categories, const char *item) {
00409     struct config_entry *entry;
00410
00411     entry = bucket_list_find_key(categories, item);
00412
00413     return (entry);
00414 }
00415
```

14.16 src/curl.c File Reference

CURL Interface.

```
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <curl/curl.h>
#include <curl/easy.h>
#include "dtsapp.h"
```

Data Structures

- struct **curl_progress**

  Allow progress monitoring.
- struct **curl_password**

  CURL Authentification callback.
- struct curl_post

  HTTP post data structure.

Functions

- int curlinit (void)

  Initilise the CURL library.
- void curlclose (void)

  Un reference CURL. This is required for each call to curlinit().
- struct curlbuf * curl_geturl (const char *def_url, struct basic_auth *bauth, curl_authcb authcb, void *auth_data)

  Fetch the URL using CURL (HTTP GET)

- struct curlbuf * curl_posturl (const char *def_url, struct basic_auth *bauth, struct curl_post *post, curl_authcb authcb, void *auth_data)

  Fetch the URL using CURL (HTTP POST)
- struct curlbuf * curl_ungzip (struct curlbuf *cbuf)

  If the buffer contains GZIP data uncompress it.
- struct basic_auth * curl_newauth (const char *user, const char *passwd)

  Create a new auth structure with initial vallues.
- struct curl_post * curl_newpost (void)

  Create a HTTP Post data structure.
- void curl_postitem (struct curl_post *post, const char *name, const char *value)

  Add a item value pair to post structure.
- char * url_escape (char *url)

  Escape and return the url.
- char * url_unescape (char *url)

  UN escape and return the url.
- void curl_setprogress (curl_progress_func cb, curl_progress_pause p_cb, curl_progress_newdata d_cb, void *data)

  Configure global progress handling.
- void curl_setauth_cb (curl_authcb auth_cb, void *data)

  Set global password callback.
- struct xml_doc * curl_buf2xml (struct curlbuf *cbuf)

  Create a XML document from from buffer (application/xml)

14.16.1 Detailed Description

CURL Interface.

Definition in file curl.c.

14.17 curl.c

```
00001
00007 #include <string.h>
00008 #include <stdint.h>
00009 #include <stdlib.h>
00010
00011 #include <curl/curl.h>
00012 #include <curl/easy.h>
00013
00014 #include "dtsapp.h"
00015
00016 static void *curl_isinit = NULL;
00017 static CURL *curl = NULL;
00018
00020 static struct curl_progress {
00022     void *data;
00024     curl_progress_func cb;
00026     curl_progress_newdata d_cb;
00028     curl_progress_pause p_cb;
00029 } *curlprogress = NULL;
00030
00032 static struct curl_password {
00034     curl_authcb authcb;
00036     void *data;
00037 } *curlpassword = NULL;
00038
00040 struct curl_post {
00042     struct curl_httppost *first;
00044     struct curl_httppost *last;
00045 };
00046
00047 static size_t bodytobuffer(void *ptr, size_t size, size_t nmemb, void *userdata) {
00048     size_t bufsize = size * nmemb;
00049     struct curlbuf *mem = (struct curlbuf *)userdata;
```

```
00050
00051     if (!(mem->body = realloc(mem->body, mem->bsize + bufsize + 1))) {
00052         return 0;
00053     }
00054     memcpy(&(mem->body[mem->bsize]), ptr, bufsize);
00055     mem->bsize += bufsize;
00056     mem->body[mem->bsize] = '\0';
00057     return bufsize;
00058 }
00059
00060 static size_t headertobuffer(void *ptr, size_t size, size_t nmemb, void *userdata) {
00061     size_t bufsize = size * nmemb;
00062     struct curlbuf *mem = (struct curlbuf *)userdata;
00063
00064     if (!(mem->header = realloc(mem->header, mem->hsize + bufsize + 1))) {
00065         return 0;
00066     }
00067     memcpy(&(mem->header[mem->hsize]), ptr, bufsize);
00068     mem->hsize += bufsize;
00069     mem->header[mem->hsize] = '\0';
00070     return bufsize;
00071 }
00072
00073 static void curlfree(void *data) {
00074     if (curl) {
00075         curl_easy_cleanup(curl);
00076         curl = NULL;
00077     }
00078     if (curlprogress) {
00079         objunref(curlprogress);
00080         curlprogress = NULL;
00081     }
00082     if (curlpassword) {
00083         objunref(curlpassword);
00084         curlpassword = NULL;
00085     }
00086 }
00087
00092 int curlinit(void) {
00093     if (curl_isinit) {
00094         return objref(curl_isinit);
00095     }
00096
00097     if (!(curl_isinit = objalloc(sizeof(void *),curlfree))) {
00098         return 0;
00099     }
00100
00101     objlock(curl_isinit);
00102     if (!(curl = curl_easy_init())) {
00103         objunlock(curl_isinit);
00104         objunref(curl_isinit);
00105         return 0;
00106     }
00107
00108     curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0);
00109     curl_easy_setopt(curl, CURLOPT_NOSIGNAL, 1);
00110     curl_easy_setopt(curl, CURLOPT_COOKIEFILE, "");
00111
00112     curl_easy_setopt(curl, CURLOPT_USERAGENT, "libcurl-agent/1.0 [Distro Solutions]");
00113
00114     curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, bodytobuffer);
00115     curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, headertobuffer);
00116     objunlock(curl_isinit);
00117     return 1;
00118 }
00119
00122 void curlclose(void) {
00123     objunref(curl_isinit);
00124     curl_isinit = NULL;
00125 }
00126
00127 static void emptybuffer(void *data) {
00128     struct curlbuf *writebuf = data;
00129
00130     if (!writebuf) {
00131         return;
00132     }
00133
00134     if (writebuf->body) {
00135         free(writebuf->body);
00136     }
00137
00138     if (writebuf->header) {
00139         free(writebuf->header);
00140     }
00141
00142     writebuf->body = NULL;
```

```
00143     writebuf->header = NULL;
00144     writebuf->bsize = 0;
00145     writebuf->hsize = 0;
00146 }
00147
00148 static struct curlbuf *curl_sendurl(const char *def_url, struct
      basic_auth *bauth, struct curl_post *post, curl_authcb authcb_in,void *
      auth_data_in) {
00149     long res;
00150     int i = 0;
00151     struct basic_auth *auth = bauth;
00152     struct curlbuf *writebuf;
00153     char userpass[64];
00154     char *url;
00155     void *p_data = NULL;
00156     curl_authcb authcb = authcb_in;
00157     void *auth_data = auth_data_in;
00158     /*    char buffer[1024];
00159         struct curl_slist *cookies, *nc;*/
00160
00161     if (!curlinit()) {
00162         return NULL;
00163     }
00164
00165     if (!(writebuf = objalloc(sizeof(*writebuf), emptybuffer))) {
00166         objunref(curl_isinit);
00167         return NULL;
00168     }
00169
00170     objlock(curl_isinit);
00171     curl_easy_setopt(curl, CURLOPT_URL, def_url);
00172     /*    curl_easy_setopt(curl, CURLOPT_ERRORBUFFER, buffer);*/
00173
00174     curl_easy_setopt(curl, CURLOPT_WRITEDATA, writebuf);
00175     curl_easy_setopt(curl, CURLOPT_WRITEHEADER, writebuf);
00176
00177     if (post) {
00178         objlock(post);
00179         curl_easy_setopt(curl, CURLOPT_HTTPPOST, post->first);
00180     }
00181
00182     if (auth && auth->user && auth->passwd) {
00183         snprintf(userpass, 63, "%s:%s", auth->user, auth->passwd);
00184         curl_easy_setopt(curl, CURLOPT_USERPWD, userpass);
00185         i++;
00186     } else if (!auth) {
00187         auth = curl_newauth(NULL, NULL);
00188     }
00189
00190     if (curlprogress && ((p_data = curlprogress->d_cb(curlprogress->data)))) {
00191         curl_easy_setopt(curl, CURLOPT_NOPROGRESS, 0);
00192         curl_easy_setopt(curl, CURLOPT_PROGRESSFUNCTION, curlprogress->cb);
00193         curl_easy_setopt(curl, CURLOPT_PROGRESSDATA, p_data);
00194     }
00195
00196     if (curlpassword && !authcb) {
00197         authcb = curlpassword->authcb;
00198         auth_data = curlpassword->data;
00199     }
00200
00201     do {
00202         if (!(res = curl_easy_perform(curl))) {
00203             curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &res);
00204             switch (res) {
00205                 /*needs auth*/
00206                 case 401:
00207                     if (curlprogress && curlprogress->p_cb) {
00208                         curlprogress->p_cb(p_data, 1);
00209                     }
00210                     if ((authcb) && ((auth = authcb((auth) ? auth->user : "", (auth) ? auth->
      passwd : "", auth_data)))) {
00211                         snprintf(userpass, 63, "%s:%s", auth->user, auth->
      passwd);
00212                         curl_easy_setopt(curl, CURLOPT_USERPWD, userpass);
00213                         emptybuffer(writebuf);
00214                     } else {
00215                         i=3;
00216                     }
00217
00218                     if (curlprogress && curlprogress->p_cb) {
00219                         curlprogress->p_cb(p_data, 0);
00220                     }
00221                     break;
00222                 /*not found*/
00223                 case 300:
00224                     i=3;
00225                     break;
```

```
00226                 /*redirect*/
00227                 case 301:
00228                     curl_easy_getinfo(curl,CURLINFO_REDIRECT_URL, &url);
00229                     curl_easy_setopt(curl, CURLOPT_URL, url);
00230                     emptybuffer(writebuf);
00231                     i--;
00232                     break;
00233                 /*ok*/
00234                 case 200:
00235                     curl_easy_getinfo(curl, CURLINFO_CONTENT_TYPE, &writebuf->
      c_type);
00236                     break;
00237             }
00238         }
00239         i++;
00240     } while ((res != 200) && (i < 3));
00241
00242     /*    curl_easy_getinfo(curl, CURLINFO_COOKIELIST, &cookies);
00243         for(nc = cookies; nc; nc=nc->next) {
00244             printf("%s\n", nc->data);
00245         }*/
00246
00247     if (!bauth) {
00248         objunref(auth);
00249     }
00250
00251     if (post) {
00252         objunlock(post);
00253         objunref(post);
00254     }
00255
00256     if (curlprogress && curlprogress->p_cb) {
00257         curlprogress->p_cb(p_data, -1);
00258     }
00259
00260     if (p_data) {
00261         objunref(p_data);
00262     }
00263
00264     objunlock(curl_isinit);
00265     objunref(curl_isinit);
00266     return writebuf;
00267 }
00268
00276 struct curlbuf *curl_geturl(const char *def_url, struct
      basic_auth *bauth, curl_authcb authcb,void *auth_data) {
00277     return curl_sendurl(def_url, bauth, NULL, authcb, auth_data);
00278 }
00279
00288 struct curlbuf *curl_posturl(const char *def_url, struct
      basic_auth *bauth, struct curl_post *post, curl_authcb authcb,void *auth_data
      ) {
00289     return curl_sendurl(def_url, bauth, post, authcb, auth_data);
00290 }
00291
00295 struct curlbuf *curl_ungzip(struct curlbuf *cbuf) {
00296     uint8_t *gzbuf;
00297     uint32_t len;
00298
00299     if (is_gzip((uint8_t *)cbuf->body, cbuf->bsize) &&
00300             ((gzbuf = gzinflatebuf((uint8_t *)cbuf->body, cbuf->
      bsize, &len)))) {
00301         free(cbuf->body);
00302         cbuf->body = gzbuf;
00303         cbuf->bsize = len;
00304     }
00305     return cbuf;
00306 }
00307
00308 static void curl_freeauth(void *data) {
00309     struct basic_auth *bauth = (struct basic_auth *)data;
00310     if (!bauth) {
00311         return;
00312     }
00313     if (bauth->user) {
00314         memset((void *)bauth->user, 0, strlen(bauth->user));
00315         free((void *)bauth->user);
00316     }
00317     if (bauth->passwd) {
00318         memset((void *)bauth->passwd, 0, strlen(bauth->passwd));
00319         free((void *)bauth->passwd);
00320     }
00321 }
00322
00328 struct basic_auth *curl_newauth(const char *user, const char *
      passwd) {
00329     struct basic_auth *bauth;
```

```
00330
00331     if (!(bauth = (struct basic_auth *)objalloc(sizeof(*bauth), curl_freeauth))) {
00332         return NULL;
00333     }
00334     if (user) {
00335         bauth->user = strdup(user);
00336     } else {
00337         bauth->user = strdup("");
00338     }
00339     if (passwd) {
00340         bauth->passwd = strdup(passwd);
00341     } else {
00342         bauth->passwd = strdup("");
00343     }
00344     return bauth;
00345 }
00346
00347 static void free_post(void *data) {
00348     struct curl_post *post = data;
00349     if (post->first) {
00350         curl_formfree(post->first);
00351     }
00352 }
00353
00356 extern struct curl_post *curl_newpost(void) {
00357     struct curl_post *post;
00358     if (!(post = objalloc(sizeof(*post), free_post))) {
00359         return NULL;
00360     }
00361     post->first = NULL;
00362     post->last = NULL;
00363     return post;
00364 }
00365
00370 void curl_postitem(struct curl_post *post, const char *name, const char *value) {
00371     if (!name || !value) {
00372         return;
00373     }
00374     objlock(post);
00375     curl_formadd(&post->first, &post->last,
00376         CURLFORM_COPYNAME, name,
00377         CURLFORM_COPYCONTENTS, value,
00378         CURLFORM_END);
00379     objunlock(post);
00380 }
00381
00385 extern char *url_escape(char *url) {
00386     char *esc;
00387     char *ret = NULL;
00388
00389     if (!curlinit()) {
00390         return NULL;
00391     }
00392
00393     objlock(curl_isinit);
00394     esc = curl_easy_escape(curl, url, 0);
00395     if (esc) {
00396         ret = strdup(esc);
00397     }
00398     curl_free(esc);
00399     objunlock(curl_isinit);
00400     objunref(curl_isinit);
00401     return ret;
00402 }
00403
00407 extern char *url_unescape(char *url) {
00408     char *uesc;
00409     char *ret = NULL;
00410
00411     if (!curlinit()) {
00412         return NULL;
00413     }
00414
00415     objlock(curl_isinit);
00416     uesc = curl_easy_unescape(curl, url, 0, 0);
00417     if (uesc) {
00418         ret = strdup(uesc);
00419     }
00420     curl_free(uesc);
00421     objunlock(curl_isinit);
00422     objunref(curl_isinit);
00423     return ret;
00424 }
00425
00426 static void free_progress(void *data) {
00427     struct curl_progress *prg = data;
00428     if (prg->data) {
```

```
00429         objunref(prg->data);
00430     }
00431 }
00432
00442 void curl_setprogress(curl_progress_func cb,
      curl_progress_pause p_cb, curl_progress_newdata d_cb, void *data) {
00443     if (curlprogress) {
00444         objunref(curlprogress);
00445         curlprogress = NULL;
00446     }
00447
00448     if (!(curlprogress = objalloc(sizeof(*curlprogress), free_progress))) {
00449         return;
00450     }
00451     curlprogress->cb = cb;
00452     curlprogress->d_cb = d_cb;
00453     curlprogress->p_cb = p_cb;
00454     if (data && objref(data)) {
00455         curlprogress->data = data;
00456     }
00457 }
00458
00459 static void free_curlpassword(void *data) {
00460     struct curl_password *cpwd = data;
00461     if (cpwd->data) {
00462         objunref(cpwd->data);
00463     }
00464 }
00465
00470 void curl_setauth_cb(curl_authcb auth_cb, void *data) {
00471     if (curlpassword) {
00472         objunref(curlpassword);
00473         curlpassword = NULL;
00474     }
00475
00476     if (!(curlpassword = objalloc(sizeof(*curlpassword), free_curlpassword))) {
00477         return;
00478     }
00479
00480     curlpassword->authcb = auth_cb;
00481     if (data && objref(data)) {
00482         curlpassword->data = data;
00483     }
00484 }
00485
00489 extern struct xml_doc *curl_buf2xml(struct curlbuf *cbuf) {
00490     struct xml_doc *xmldoc = NULL;
00491
00492     if (cbuf && cbuf->c_type && !strcmp("application/xml", cbuf->c_type)) {
00493         curl_ungzip(cbuf);
00494         xmldoc = xml_loadbuf(cbuf->body, cbuf->bsize, 1);
00495     }
00496     return xmldoc;
00497 }
00498
```

14.18 src/fileutil.c File Reference

File utilities to test files (fstat)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fcntl.h>
#include <ctype.h>
#include <grp.h>
```

Functions

- int is_file (const char *path)

  Determine if a file exists.
- int is_dir (const char *path)

  Determine if a path is a directory.
- int is_exec (const char *path)

  Determine if a file is executable.
- int mk_dir (const char *dir, mode_t mode, uid_t user, gid_t group)

  Create a directory.

14.18.1 Detailed Description

File utilities to test files (fstat)

Definition in file fileutil.c.

14.19 fileutil.c

```
00001 /*
00002     Distrotech Solutions wxWidgets Gui Interface
00003     Copyright (C) 2013 Gregory Hinton Nietsky <gregory@distrotech.co.za>
00004
00005     This program is free software: you can redistribute it and/or modify
00006     it under the terms of the GNU General Public License as published by
00007     the Free Software Foundation, either version 3 of the License, or
00008     (at your option) any later version.
00009
00010     This program is distributed in the hope that it will be useful,
00011     but WITHOUT ANY WARRANTY; without even the implied warranty of
00012     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013     GNU General Public License for more details.
00014
00015     You should have received a copy of the GNU General Public License
00016     along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00025 #include <sys/types.h>
00026 #include <sys/stat.h>
00027 #include <unistd.h>
00028 #include <errno.h>
00029 #include <string.h>
00030 #include <stdio.h>
00031 #include <stdlib.h>
00032 #include <math.h>
00033 #include <fcntl.h>
00034 #include <ctype.h>
00035 #ifdef __WIN32
00036 #else
00037 #include <grp.h>
00038 #endif
00039
00043 extern int is_file(const char *path) {
00044     struct stat sr;
00045     if (!stat(path, &sr)) {
00046         return 1;
00047     } else {
00048         return 0;
00049     }
00050 }
00051
00055 extern int is_dir(const char *path) {
00056     struct stat sr;
00057     if (!stat(path, &sr) && S_ISDIR(sr.st_mode)) {
00058         return 1;
00059     } else {
00060         return 0;
00061     }
00062 }
00063
00067 extern int is_exec(const char *path) {
00068     struct stat sr;
00069     if (!stat(path, &sr) && (S_IXUSR & sr.st_mode)) {
```

```
00070         return 1;
00071     } else {
00072         return 0;
00073     }
00074 }
00075
00084 #ifdef __WIN32
00085 extern int mk_dir(const char *dir) {
00086 #else
00087 extern int mk_dir(const char *dir, mode_t mode, uid_t user, gid_t group) {
00088 #endif
00089     struct stat sr;
00090
00091 #ifdef __WIN32
00092     if ((stat(dir, &sr) && (errno == ENOENT)) && !mkdir(dir)) {
00093 #else
00094     if ((stat(dir, &sr) && (errno == ENOENT)) && !mkdir(dir, mode) && !chown(dir, user, group)) {
00095 #endif
00096         return 0;
00097     }
00098     return -1;
00099 }
00100
```

14.20 src/include/dtsapp.h File Reference

DTS Application library API Include file.

```
#include <signal.h>
#include <arpa/inet.h>
#include <linux/un.h>
```

Data Structures

- union sockstruct

  Socket union describing all address types.
- struct fwsocket

  Socket data structure.
- struct config_entry

  Configuration category entry.
- struct zobj

  Zlib buffer used for compression and decompression.
- struct ifinfo

  Data structure containing interface information.
- struct framework_core

  Application framework data.
- struct xml_attr

  XML attribute name value pair.
- struct xml_node

  Reference to a XML Node.
- struct ldap_rdn

  LDAP Relative distingushed name linked list.
- struct ldap_attrval

  LDAP attribute value.
- struct ldap_attr

  LDAP attirbute.
- struct ldap_entry

  LDAP entry.
- struct ldap_results

LDAP results.

- struct basic_auth

  Basic authentification structure.

- struct curlbuf

  Buffer containing the result of a curl transaction.

Macros

- #define RAD_AUTH_HDR_LEN 20

  Authentification header length.

- #define RAD_AUTH_PACKET_LEN 4096

  Auth packet length.

- #define RAD_AUTH_TOKEN_LEN 16

  Auth token length.

- #define RAD_MAX_PASS_LEN 128

  Auth max password length.

- #define RAD_ATTR_USER_NAME 1 /∗string∗/

  Radius attribute username.

- #define RAD_ATTR_USER_PASSWORD 2 /∗passwd∗/

  Radius attribute password.

- #define RAD_ATTR_NAS_IP_ADDR 4 /∗ip∗/

  Radius attribute server IP.

- #define RAD_ATTR_NAS_PORT 5 /∗int∗/

  Radius attribute server port.

- #define RAD_ATTR_SERVICE_TYPE 6 /∗int∗/

  Radius attribute service type.

- #define RAD_ATTR_ACCTID 44

  Radius attribute account id.

- #define RAD_ATTR_PORT_TYPE 61 /∗int∗/

  Radius attribute port type.

- #define RAD_ATTR_EAP 79 /∗oct∗/

  Radius attribute EAP.

- #define RAD_ATTR_MESSAGE 80 /∗oct∗/

  Radius attribute message.

- #define JHASH_INITVAL 0xdeadbeef

  Default init value for hash function
  easter egg copied from <linux/jhash.h>

- #define jenhash(key, length, initval) hashlittle(key, length, (initval) ? initval : JHASH_INITVAL);

  Define jenhash as hashlittle on big endian it should be hashbig.

- #define clearflag(obj, flag)

  Atomically clear a flag in the flags field of a referenced object.

- #define setflag(obj, flag)

  Atomically set a flag in the flags field of a referenced object.

- #define testflag(obj, flag) (objlock(obj) | (obj->flags & flag) | objunlock(obj))

  Atomically test a flag in the flags field of a referenced object.

- #define FRAMEWORK_MAIN(progname, name, email, www, year, runfile, flags, sighfunc)

  A macro to replace main() with initilization and daemonization code.

- #define ALLOC_CONST(const_var, val)

  Macro to assign values to char const.

- #define DTS_OJBREF_CLASS(classtype)

  Add this macro to a C++ class to add refobj support.

Typedefs

- typedef struct ssldata ssldata

  Forward decleration of structure.
- typedef struct natmap natmap

  Forward decleration of structure.
- typedef struct radius_packet radius_packet

  Forward decleration of structure.
- typedef struct nfq_queue nfq_queue

  Forward decleration of structure.
- typedef struct nfq_data nfq_data

  Forward decleration of structure.
- typedef struct nfct_struct nfct_struct

  Forward decleration of structure.
- typedef struct nfqnl_msg_packet_hdr nfqnl_msg_packet_hdr

  Forward decleration of structure.
- typedef int(∗ frameworkfunc)(int, char ∗∗)

  Framework callback function.
- typedef void(∗ syssighandler)(int, siginfo_t ∗, void ∗)

  Callback to user supplied signal handler.
- typedef void(∗ threadcleanup)(void ∗)

  Function called after thread termination.
- typedef void ∗(∗ threadfunc)(void ∗)

  Thread function.
- typedef int(∗ threadsighandler)(int, void ∗)

  Thread signal handler function.
- typedef void(∗ socketrecv)(struct fwsocket ∗, void ∗)

  Callback function to register with a socket that will be called when there is data available.
- typedef void(∗ objdestroy)(void ∗)

  Callback used to clean data of a reference object when it is to be freed.
- typedef int32_t(∗ blisthash)(const void ∗, int)

  Callback used to calculate the hash of a structure.
- typedef void(∗ blist_cb)(void ∗, void ∗)

  This callback is run on each entry in a list.
- typedef void(∗ config_filecb)(struct bucket_list ∗, const char ∗, const char ∗)

  Calback used when processing config files.
- typedef void(∗ config_catcb)(struct bucket_list ∗, const char ∗)

  Calback used when processing a category.
- typedef void(∗ config_entrycb)(const char ∗, const char ∗)

  Callback used when processing a entry.
- typedef uint32_t(∗ nfqueue_cb)(struct nfq_data ∗, struct nfqnl_msg_packet_hdr ∗, char ∗, uint32_t, void ∗, uint32_t ∗, void ∗∗)
- typedef void(∗ radius_cb)(struct radius_packet ∗, void ∗)

  Callback to call when response arrives.
- typedef struct xml_node xml_node

  Forward decleration of structure.
- typedef struct xml_search xml_search

  Forward decleration of structure.
- typedef struct xml_doc xml_doc

  Forward decleration of structure.
- typedef struct xslt_doc xslt_doc

Forward decleration of structure.

- typedef struct ldap_conn ldap_conn

  Forward decleration of structure.
- typedef struct ldap_modify ldap_modify

  Forward decleration of structure.
- typedef struct ldap_add ldap_add

  Forward decleration of structure.
- typedef struct curl_post curl_post

  Forward decleration of structure.
- typedef struct basic_auth *(* curl_authcb)(const char *, const char *, void *)

  Callback to set the authentification ie on error 401.
- typedef int(* curl_progress_func)(void *, double, double, double, double)

  CURL callback function called when there is progress (CURLOPT_PROGRESSFUNCTION).
- typedef void(* curl_progress_pause)(void *, int)

  Callback function to control the progress bar.
- typedef void *(* curl_progress_newdata)(void *)

  Create a new progress data structure.

Enumerations

- enum sock_flags {
  SOCK_FLAG_BIND = 1 << 0, SOCK_FLAG_CLOSE = 1 << 1, SOCK_FLAG_SSL = 1 << 2, SOCK_FLAG_UNIX = 1 << 3,
  SOCK_FLAG_MCAST = 1 << 4 }

  Socket flags controling a socket.
- enum thread_option_flags { THREAD_OPTION_CANCEL = 1 << 0, THREAD_OPTION_JOINABLE = 1 << 1, THREAD_OPTION_RETURN = 1 << 2 }

  Options supplied to framework_mkthread all defaults are unset.
- enum framework_flags { FRAMEWORK_FLAG_DAEMON = 1 << 0, FRAMEWORK_FLAG_NOGNU = 1 << 1, FRAMEWORK_FLAG_DAEMONLOCK = 1 << 2 }

  Application control flags.
- enum RADIUS_CODE {
  RAD_CODE_AUTHREQUEST = 1, RAD_CODE_AUTHACCEPT = 2, RAD_CODE_AUTHREJECT = 3, RAD_CODE_ACCTREQUEST = 4,
  RAD_CODE_ACCTRESPONSE = 5, RAD_CODE_AUTHCHALLENGE = 11 }

  Radius packet codes.
- enum ldap_starttls { LDAP_STARTTLS_NONE, LDAP_STARTTLS_ATTEMPT, LDAP_STARTTLS_ENFORCE }

  SSL connection requirements.
- enum ldap_attrtype { LDAP_ATTRTYPE_CHAR, LDAP_ATTRTYPE_B64, LDAP_ATTRTYPE_OCTET }

  LDAP attribute types.

Functions

- void framework_mkcore (char *progname, char *name, char *email, char *web, int year, char *runfile, int flags, syssighandler sigfunc)

  Initilise application data structure and return a reference.
- int framework_init (int argc, char *argv[], frameworkfunc callback)

  Initilise the application daemonise and join the manager thread.
- void printgnu (const char *pname, int year, const char *dev, const char *email, const char *www)

  Print a brief GNU copyright notice on console.
- void daemonize ()

Daemonise the application using fork/exit.

- int lockpidfile (const char *runfile)

  Lock the run file in the framework application info.
- struct thread_pvt * framework_mkthread (threadfunc, threadcleanup, threadsighandler, void *data, int flags)

  create a thread result must be unreferenced
- struct fwsocket * unixsocket_server (const char *sock, int protocol, int mask, socketrecv read, void *data)

  Create and run UNIX server socket thread.
- struct fwsocket * unixsocket_client (const char *sock, int protocol, socketrecv read, void *data)

  Create a client thread on the socket.
- int framework_threadok (void)

  let threads check there status.
- int startthreads (void)

  Initialise the threadlist and start manager thread.
- void stopthreads (int join)

  Signal manager to stop and cancel all running threads.
- int thread_signal (int sig)

  Handle signal if its for me.
- int objlock (void *data)

  Lock the reference.
- int objtrylock (void *data)

  Try lock a reference.
- int objunlock (void *data)

  Unlock a reference.
- int objcnt (void *data)

  Return current reference count.
- int objsize (void *data)

  Size requested for data.
- int objunref (void *data)

  Drop reference held.
- int objref (void *data)

  Reference a object.
- void * objalloc (int size, objdestroy)

  Allocate a referenced lockable object.
- void * objchar (const char *orig)

  Return a reference to copy of a buffer.
- void * create_bucketlist (int bitmask, blisthash hash_function)
- int addtobucket (struct bucket_list *blist, void *data)

  Add a reference to the bucketlist.
- void remove_bucket_item (struct bucket_list *blist, void *data)

  Remove and unreference a item from the list.
- int bucket_list_cnt (struct bucket_list *blist)

  Return number of items in the list.
- void * bucket_list_find_key (struct bucket_list *list, const void *key)

  Find and return a reference to a item matching supplied key.
- void bucketlist_callback (struct bucket_list *blist, blist_cb callback, void *data2)

  Run a callback function on all items in the list.
- struct bucket_loop * init_bucket_loop (struct bucket_list *blist)

  Create a bucket list iterator to safely iterate the list.
- void * next_bucket_loop (struct bucket_loop *bloop)

  Return a reference to the next item in the list this could be the first item.
- void remove_bucket_loop (struct bucket_loop *bloop)

Safely remove a item from a list while iterating in a loop.

- uint32_t hashlittle (const void *key, size_t length, uint32_t initval)

  hash a variable-length key into a 32-bit value (Little Endian)
- void seedrand (void)

  Seed openssl random number generator.
- int genrand (void *buf, int len)

  Generate random sequence.
- void sha512sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA2-512 hash.
- void sha256sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA2-256 hash.
- void sha1sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA1 hash.
- void md5sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the MD5 hash.
- void sha512sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA2-512 hash accross 2 data chunks.
- void sha256sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA2-256 hash accross 2 data chunks.
- void sha1sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA1 hash accross 2 data chunks.
- void md5sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the MD5 hash accross 2 data chunks.
- int sha512cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA2-512 hashes.
- int sha256cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA2-256 hashes.
- int sha1cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA1 hashes.
- int md5cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two md5 hashes.
- void sha512hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA2-512.
- void sha256hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA2-256.
- void sha1hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA1.
- void md5hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) MD5.
- int strlenzero (const char *str)

  Check if a string is zero length.
- char * ltrim (char *str)

  Trim white space at the begining of a string.
- char * rtrim (const char *str)

Trim white space at the end of a string.

- char * trim (const char *str)

  Trim whitesapce from the beggining and end of a string.
- uint64_t tvtontp64 (struct timeval *tv)

  Convert a timeval struct to 64bit NTP time.
- uint16_t checksum (const void *data, int len)

  Obtain the checksum for a buffer.
- uint16_t checksum_add (const uint16_t checksum, const void *data, int len)

  Obtain the checksum for a buffer adding a checksum.
- uint16_t verifysum (const void *data, int len, const uint16_t check)

  Verify a checksum.
- struct zobj * zcompress (uint8_t *buff, uint16_t len, uint8_t level)

  Allocate a buffer and return it with compressed data.
- void zuncompress (struct zobj *buff, uint8_t *obuff)

  Uncompress zobj buffer to buffer.
- uint8_t * gzinflatebuf (uint8_t *buf_in, int buf_size, uint32_t *len)

  Ungzip a buffer.
- int is_gzip (uint8_t *buf, int buf_size)

  check a buffer if it contains gzip magic
- void touch (const char *filename, uid_t user, gid_t group)

  Create a file and set user and group.
- char * b64enc (const char *message, int nonl)

  Base 64 encode a string.
- char * b64enc_buf (const char *message, uint32_t len, int nonl)

  Base 64 encode a buffer.
- struct fwsocket * make_socket (int family, int type, int proto, void *ssl)

  Allocate a socket structure and return reference.
- struct fwsocket * accept_socket (struct fwsocket *sock)

  Create and return a socket structure from accept()
- struct fwsocket * sockconnect (int family, int stype, int proto, const char *ipaddr, const char *port, void *ssl)

  Generic client socket.
- struct fwsocket * udpconnect (const char *ipaddr, const char *port, void *ssl)

  UDP Socket client.
- struct fwsocket * tcpconnect (const char *ipaddr, const char *port, void *ssl)

  TCP Socket client.
- struct fwsocket * sockbind (int family, int stype, int proto, const char *ipaddr, const char *port, void *ssl, int backlog)

  Generic server socket.
- struct fwsocket * udpbind (const char *ipaddr, const char *port, void *ssl)

  UDP server socket.
- struct fwsocket * tcpbind (const char *ipaddr, const char *port, void *ssl, int backlog)

  Generic server socket.
- void close_socket (struct fwsocket *sock)

  Mark the socket for closure and release the reference.
- int score_ipv4 (struct sockaddr_in *sa4, char *ipaddr, int iplen)

  Return a score for a IPv4 addrress.
- int score_ipv6 (struct sockaddr_in6 *sa6, char *ipaddr, int iplen)

  Return a score for a IPv6 addrress.
- int inet_lookup (int family, const char *host, void *addr, socklen_t len)

  Perform DNS lookup on a host/ip retun the IP address.
- void socketclient (struct fwsocket *sock, void *data, socketrecv read, threadcleanup cleanup)

Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.

- void socketserver (struct fwsocket *sock, socketrecv connectfunc, socketrecv acceptfunc, threadcleanup cleanup, void *data)

  Create a server thread with a socket that has been created with sockbind udpbind or tcpbind.
- struct fwsocket * mcast_socket (const char *iface, int family, const char *mcastip, const char *port, int flags)

  Create a multicast socket.
- const char * sockaddr2ip (union sockstruct *addr, char *buf, int len)

  Return the ip address of a sockstruct addr.
- int checkipv6mask (const char *ipaddr, const char *network, uint8_t bits)

  Check if ipaddr is in a network.
- void ipv4tcpchecksum (uint8_t *pkt)

  Update the TCP checksum of a IPv4 packet.
- void ipv4udpchecksum (uint8_t *pkt)

  Update the UDP checksum of a IPv4 packet.
- void ipv4icmpchecksum (uint8_t *pkt)

  Set the checksup of a IPv4 ICMP packet.
- void ipv4checksum (uint8_t *pkt)

  Set the checksup of a IPv4 Packet.
- int packetchecksumv4 (uint8_t *pkt)

  Update the checksum of a IPv4 packet.
- int packetchecksumv6 (uint8_t *pkt)

  Prototype to check checksup on packet.
- int packetchecksum (uint8_t *pkt)

  Generic IPv4 and IPv6 Checksum.
- void rfc6296_map (struct natmap *map, struct in6_addr *ipaddr, int out)

  Lookup and process a NAT transform as per RFC 6296.
- int rfc6296_map_add (char *intaddr, char *extaddr)

  Calculate and add a NAT map.
- const char * cidrtosn (int bitlen, char *buf, int size)

  Return the dotted quad notation subnet mask from a CIDR.
- const char * getnetaddr (const char *ipaddr, int cidr, char *buf, int size)

  Return the network address.
- const char * getbcaddr (const char *ipaddr, int cidr, char *buf, int size)

  Return broadcast address.
- const char * getfirstaddr (const char *ipaddr, int cidr, char *buf, int size)

  Get the first usable address.
- const char * getlastaddr (const char *ipaddr, int cidr, char *buf, int size)

  Get the last usable address.
- uint32_t cidrcnt (int bitlen)

  Return the number of IP addresses in a given bitmask.
- int reservedip (const char *ipaddr)

  Check IP against list of reserved IP's.
- char * ipv6to4prefix (const char *ipaddr)

  Return IPv6 to IPv4 Prefix fot the address.
- int check_ipv4 (const char *ip, int cidr, const char *test)

  Check if a IP address is in a network.
- void mcast4_ip (struct in_addr *addr)

  Randomally assign a SSM Multicast address.
- void mcast6_ip (struct in6_addr *addr)

  Randomally assign a SSM Multicast address.
  param addr Ip address structure to fill out.

- struct nfq_queue ∗ nfqueue_attach (uint16_t pf, uint16_t num, uint8_t mode, uint32_t range, nfqueue_cb cb, void ∗data)
- uint16_t snprintf_pkt (struct nfq_data ∗tb, struct nfqnl_msg_packet_hdr ∗ph, uint8_t ∗pkt, char ∗buff, uint16_t len)
- struct nf_conntrack ∗ nf_ctrack_buildct (uint8_t ∗pkt)
- uint8_t nf_ctrack_delete (uint8_t ∗pkt)
- uint8_t nf_ctrack_nat (uint8_t ∗pkt, uint32_t addr, uint16_t port, uint8_t dnat)
- void nf_ctrack_dump (void)
- struct nfct_struct ∗ nf_ctrack_trace (void)
- void nf_ctrack_endtrace (struct nfct_struct ∗nfct)
- uint8_t nf_ctrack_init (void)
- void nf_ctrack_close (void)
- int delete_kernvlan (char ∗ifname, int vid)

  Delete a VLAN.
- int create_kernvlan (char ∗ifname, unsigned short vid)

  Create a VLAN on a interface.
- int delete_kernmac (char ∗macdev)

  Delete Kernel MAC VLAN.
- int create_kernmac (char ∗ifname, char ∗macdev, unsigned char ∗mac)

  Create a kernal MAC VLAN.
- int interface_bind (char ∗iface, int protocol)

  Bind to device fd may be a existing socket.
- void randhwaddr (unsigned char ∗addr)

  create random MAC address
- int create_tun (const char ∗ifname, const unsigned char ∗hwaddr, int flags)

  Create a tunnel device.
- int ifrename (const char ∗oldname, const char ∗newname)

  Rename interface helper.
- int ifdown (const char ∗ifname, int flags)

  Set interface down.
- int ifup (const char ∗ifname, int flags)

  Set interface up.
- int ifhwaddr (const char ∗ifname, unsigned char ∗hwaddr)

  Get MAC addr for interface.
- int set_interface_flags (int ifindex, int set, int clear)

  Alter interface flags.
- int get_iface_index (const char ∗ifname)

  Get the netlink interface for a named interface.
- int set_interface_addr (int ifindex, const unsigned char ∗hwaddr)

  Set interface MAC addr.
- int set_interface_name (int ifindex, const char ∗name)

  Rename interface.
- int set_interface_ipaddr (char ∗ifname, char ∗ipaddr)

  Set IP addr on interface.
- int get_ip6_addrprefix (const char ∗iface, unsigned char ∗prefix)

  Generate Unique Local IPv6 Unicast Addresses RFC 4193.
- void eui48to64 (unsigned char ∗mac48, unsigned char ∗eui64)

  Generate IPv6 address from mac address.
- void closenetlink (void)

  Close netlink socket on application termination.
- const char ∗ get_ifipaddr (const char ∗iface, int family)

  Find best IP adress for a interface.

- void addradattrint (struct radius_packet *packet, char type, unsigned int val)

  Add a integer attribute too the packet.
- void addradattrip (struct radius_packet *packet, char type, char *ipaddr)

  Add a integer attribute too the packet.
- void addradattrstr (struct radius_packet *packet, char type, char *str)

  Add a integer attribute too the packet.
- struct radius_packet * new_radpacket (unsigned char code)

  Create a new radius packet.
- int send_radpacket (struct radius_packet *packet, const char *userpass, radius_cb read_cb, void *cb_data)

  Send radius packet.
- void add_radserver (const char *ipaddr, const char *auth, const char *acct, const char *secret, int timeout)

  Add new radius server to list of servers.
- unsigned char * radius_attr_first (struct radius_packet *packet)

  Return first packet attribute.
- unsigned char * radius_attr_next (struct radius_packet *packet, unsigned char *attr)

  Return next packet attribute.
- void sslstartup (void)

  Initialise SSL support this should be called at startup.
- void * tlsv1_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for TLSv1.
- void * sslv2_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for SSLv2 (If available)
- void * sslv3_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for SSLv3.
- void * dtlsv1_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for DTLSv1.
- int socketread (struct fwsocket *sock, void *buf, int num)

  Read from a socket into a buffer.
- int socketwrite (struct fwsocket *sock, const void *buf, int num)

  Write a buffer to a socket.
- int socketread_d (struct fwsocket *sock, void *buf, int num, union sockstruct *addr)

  Read from a socket into a buffer.
- int socketwrite_d (struct fwsocket *sock, const void *buf, int num, union sockstruct *addr)

  Write a buffer to a socket.
- void ssl_shutdown (void *ssl, int sock)

  Shutdown the SSL connection.
- void tlsaccept (struct fwsocket *sock, struct ssldata *orig)

  Create SSL session for new connection.
- struct fwsocket * dtls_listenssl (struct fwsocket *sock)

  Implementation of "listen" for DTLSv1.
- void startsslclient (struct fwsocket *sock)

  Start SSL on a client socket.
- void unrefconfigfiles (void)

  Empty out and unreference config files.
- int process_config (const char *configname, const char *configfile)

  Process a configfile into buckets.
- struct bucket_loop * get_category_loop (const char *configname)

  Return a bucket loop to allow iterating over categories.
- struct bucket_list * get_category_next (struct bucket_loop *cloop, char *name, int len)

  Iterate through categories returning the entries bucket.
- struct bucket_list * get_config_category (const char *configname, const char *category)

Return a single category.

- struct config_entry * get_config_entry (struct bucket_list *categories, const char *item)

  Find the entry in a config file.
- void config_file_callback (config_filecb file_cb)

  Callback wrapper to iterate over all configfiles calling a callback on each file.
- void config_cat_callback (struct bucket_list *categories, config_catcb entry_cb)

  Callback wrapper that iterates through categories calling a callback on each category.
- void config_entry_callback (struct bucket_list *entries, config_entrycb entry_cb)

  Callback Wraper that iterates through all items calling a callback for each item.
- struct xml_doc * xml_loaddoc (const char *docfile, int validate)

  Load a XML file into XML document and return reference.
- struct xml_doc * xml_loadbuf (const uint8_t *buffer, uint32_t len, int validate)

  Load a buffer into XML document returning refereence.
- struct xml_node * xml_getfirstnode (struct xml_search *xpsearch, void **iter)

  Return reference to the first node optionally creating a iterator.
- struct xml_node * xml_getnextnode (void *iter)

  Return the next node.
- struct bucket_list * xml_getnodes (struct xml_search *xpsearch)

  Return reference to bucket list containing nodes.
- struct xml_search * xml_xpath (struct xml_doc *xmldata, const char *xpath, const char *attrkey)

  Return a reference to a xpath search result.
- int xml_nodecount (struct xml_search *xsearch)

  Return the number of nodes in the search path.
- struct xml_node * xml_getnode (struct xml_search *xsearch, const char *key)

  Return a node in the search matching key.
- const char * xml_getattr (struct xml_node *xnode, const char *attr)

  Return value of attribute.
- void xml_modify (struct xml_doc *xmldoc, struct xml_node *xnode, const char *value)

  Modify a XML node.
- void xml_setattr (struct xml_doc *xmldoc, struct xml_node *xnode, const char *name, const char *value)

  Modify a XML node attribute.
- struct xml_node * xml_addnode (struct xml_doc *xmldoc, const char *xpath, const char *name, const char *value, const char *attrkey, const char *keyval)

  Append a node to a path.
- void xml_appendnode (struct xml_doc *xmldoc, const char *xpath, struct xml_node *child)

  Append a node to a path.
- void xml_unlink (struct xml_node *xnode)

  Unlink a node from the document.
- void xml_delete (struct xml_node *xnode)

  Delete a node from document it is not unrefd and should be.
- char * xml_getbuffer (void *buffer)

  Return the buffer of a xml_buffer structure.
- void * xml_doctobuffer (struct xml_doc *xmldoc)

  Return a dump of a XML document.
- const char * xml_getrootname (struct xml_doc *xmldoc)

  Return the name of the root node.
- struct xml_node * xml_getrootnode (struct xml_doc *xmldoc)

  Return reference to the root node.
- void xml_savefile (struct xml_doc *xmldoc, const char *file, int format, int compress)

  Save XML document to a file.
- void xml_createpath (struct xml_doc *xmldoc, const char *xpath)

Create a path in XML document.

- void xml_init ()

  Initialise/Reference the XML library.

- void xml_close ()

  Unreference the XML library.

- struct xslt_doc * xslt_open (const char *xsltfile)

  Open a XSLT file returning reference to it.

- void xslt_addparam (struct xslt_doc *xsltdoc, const char *param, const char *value)

  Add a parameter to the XSLT document.

- void xslt_apply (struct xml_doc *xmldoc, struct xslt_doc *xsltdoc, const char *filename, int comp)

  Apply XSLT document to a XML document.

- void * xslt_apply_buffer (struct xml_doc *xmldoc, struct xslt_doc *xsltdoc)

  Apply XSLT document to a XML document returning result in buffer.

- void xslt_init ()

  Reference the XSLT parser.

- void xslt_close ()

  Release reference to XSLT parser.

- struct ldap_conn * ldap_connect (const char *uri, enum ldap_starttls starttls, int timelimit, int limit, int debug, int *err)

  Connect to a LDAP server.

- int ldap_simplebind (struct ldap_conn *ld, const char *dn, const char *passwd)

  Bind to the connection with simple bind requireing a distingushed name and password.

- int ldap_saslbind (struct ldap_conn *ld, const char *mech, const char *realm, const char *authcid, const char *passwd, const char *authzid)

  Bind to the server with SASL.

- int ldap_simplerebind (struct ldap_conn *ld, const char *initialdn, const char *initialpw, const char *base, const char *filter, const char *uidrdn, const char *uid, const char *passwd)

  Bind to LDAP connection using rebind.

- const char * ldap_errmsg (int res)

  Return LDAP error for a ldap error.

- struct ldap_results * ldap_search_sub (struct ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res,...)

  Search LDAP connection subtree.

- struct ldap_results * ldap_search_one (struct ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res,...)

  Search LDAP connection one level.

- struct ldap_results * ldap_search_base (struct ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res,...)

  Search LDAP connection base.

- void ldap_unref_entry (struct ldap_results *results, struct ldap_entry *entry)

  Remove a entry from a result.

- void ldap_unref_attr (struct ldap_entry *entry, struct ldap_attr *attr)

  Remove a attribute from a entry.

- struct ldap_entry * ldap_getentry (struct ldap_results *results, const char *dn)

  Find and return the entry from the results for a specific dn.

- struct ldap_attr * ldap_getattr (struct ldap_entry *entry, const char *attr)

  Find and return attribute in a entry.

- struct ldap_modify * ldap_modifyinit (const char *dn)

  Create a modification reference for a DN.

- int ldap_mod_del (struct ldap_modify *lmod, const char *attr,...)

  Delete values from a attribute.

- int ldap_mod_add (struct ldap_modify *lmod, const char *attr,...)

  Add values to a attribute.
- int ldap_mod_rep (struct ldap_modify *lmod, const char *attr,...)

  Replace a attribute.
- int ldap_domodify (struct ldap_conn *ld, struct ldap_modify *lmod)

  Apply the modification to the server.
- int ldap_mod_rematttr (struct ldap_conn *ldap, const char *dn, const char *attr)

  Delete a attribute from a DN.
- int ldap_mod_delattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Delete a value from a attribute in a DN.
- int ldap_mod_addattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Add a value for a attribute in a DN.
- int ldap_mod_repattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Replace the value of a attribute in a DN.
- int curlinit (void)

  Initilise the CURL library.
- void curlclose (void)

  Un reference CURL. This is required for each call to curlinit().
- struct basic_auth * curl_newauth (const char *user, const char *passwd)

  Create a new auth structure with initial vallues.
- struct curlbuf * curl_geturl (const char *def_url, struct basic_auth *bauth, curl_authcb authcb, void *data)

  Fetch the URL using CURL (HTTP GET)
- void curl_setprogress (curl_progress_func cb, curl_progress_pause p_cb, curl_progress_newdata d_cb, void *data)

  Configure global progress handling.
- void curl_setauth_cb (curl_authcb auth_cb, void *data)

  Set global password callback.
- struct curl_post * curl_newpost (void)

  Create a HTTP Post data structure.
- void curl_postitem (struct curl_post *post, const char *name, const char *item)

  Add a item value pair to post structure.
- struct curlbuf * curl_posturl (const char *def_url, struct basic_auth *bauth, struct curl_post *post, curl_authcb authcb, void *data)

  Fetch the URL using CURL (HTTP POST)
- struct curlbuf * curl_ungzip (struct curlbuf *cbuf)

  If the buffer contains GZIP data uncompress it.
- struct xml_doc * curl_buf2xml (struct curlbuf *cbuf)

  Create a XML document from from buffer (application/xml)
- char * url_escape (char *url)

  Escape and return the url.
- char * url_unescape (char *url)

  UN escape and return the url.
- int is_file (const char *path)

  Determine if a file exists.
- int is_dir (const char *path)

  Determine if a path is a directory.
- int is_exec (const char *path)

  Determine if a file is executable.
- int mk_dir (const char *dir, mode_t mode, uid_t user, gid_t group)

  Create a directory.

14.20.1 Detailed Description

DTS Application library API Include file. The library foremostly implements reference counted objects and hashed bucket lists Referenced Lockable Objects these are then used to implement simpler API's to common tasks.

Key components

INI style config file parser.
CURL wraper with support for GET/POST, authentification and progress indication.
File utilities as a wrapper arround fstat.
IP 4/6 Utilities for calculating / checking subnets and checksuming packets.
Interface API for Linux networking including libnetlink from iproute2
XML/XSLT Simplified API for reading, managing and applying transforms.
Some Application shortcuts and wrapper for main quick and dirty daemon app.
Wrappers for Linux netfilter connection tracking and packet queueing
Open LDAP API.
Basic implementation of RADIUS.
Implementation of RFC 6296.
Thread API using pthreads.
Simple implementation of UNIX Domain socket.
Various Utilities including hashing and checksum.
Z Lib Compression/Uncompression Functions.

Definition in file dtsapp.h.

14.21 dtsapp.h

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00019 /*
00020  * Acknowledgments [MD5 HMAC http://www.ietf.org/rfc/rfc2104.txt]
00021  *  Pau-Chen Cheng, Jeff Kraemer, and Michael Oehler, have provided
00022  *  useful comments on early drafts, and ran the first interoperability
00023  *  tests of this specification. Jeff and Pau-Chen kindly provided the
00024  *  sample code and test vectors that appear in the appendix.  Burt
00025  *  Kaliski, Bart Preneel, Matt Robshaw, Adi Shamir, and Paul van
00026  *  Oorschot have provided useful comments and suggestions during the
00027  *  investigation of the HMAC construction.
00028  */
00029
00030 /*
00031  * User password crypt function from the freeradius project (addattrpasswd)
00032  * Copyright (C) 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009 The FreeRADIUS Server
       Project
00033  */
00034
00058 #ifndef _INCLUDE_DTSAPP_H
00059 #define _INCLUDE_DTSAPP_H
00060
00061 #include <signal.h>
00062 #ifdef __WIN32__
00063 #include <winsock2.h>
00064 #include <ws2ipdef.h>
00065 #include <iphlpapi.h>
00066 #include <ws2tcpip.h>
00067 #else
```

```
00068 #include <arpa/inet.h>
00069 #include <linux/un.h>
00070 #endif
00071
00072 #ifdef __cplusplus
00073 extern "C" {
00074 #endif
00075
00076
00080 union sockstruct {
00082     struct sockaddr sa;
00083 #ifndef __WIN32
00084
00085     struct sockaddr_un un;
00086 #endif
00087
00088     struct sockaddr_in sa4;
00090     struct sockaddr_in6 sa6;
00092     struct sockaddr_storage ss;
00093 };
00094
00097 typedef struct ssldata ssldata;
00098
00102 enum sock_flags {
00104     SOCK_FLAG_BIND        = 1 << 0,
00106     SOCK_FLAG_CLOSE      = 1 << 1,
00108     SOCK_FLAG_SSL      = 1 << 2,
00110     SOCK_FLAG_UNIX        = 1 << 3,
00112     SOCK_FLAG_MCAST      = 1 << 4
00113 };
00114
00118 enum thread_option_flags {
00120         THREAD_OPTION_CANCEL        = 1 << 0,
00122         THREAD_OPTION_JOINABLE        = 1 << 1,
00124         THREAD_OPTION_RETURN        = 1 << 2
00125 };
00126
00127
00131 struct fwsocket {
00133     int sock;
00135     int proto;
00137     int type;
00140     enum sock_flags flags;
00143     union sockstruct addr;
00146     struct ssldata *ssl;
00148     struct fwsocket *parent;
00150     struct bucket_list *children;
00151 };
00152
00155 struct config_entry {
00157     const char *item;
00159     const char *value;
00160 };
00161
00164 struct zobj {
00166     uint8_t *buff;
00168     uint16_t olen;
00170     uint16_t zlen;
00171 };
00172
00176 struct ifinfo {
00178         int idx;
00180         const char *ifaddr;
00182         const char *ipv4addr;
00184         const char *ipv6addr;
00185 };
00186
00189 typedef struct natmap natmap;
00190
00193 typedef struct radius_packet radius_packet;
00194
00197 typedef struct nfq_queue nfq_queue;
00198
00201 typedef struct nfq_data nfq_data;
00202
00205 typedef struct nfct_struct nfct_struct;
00206
00209 typedef struct nfqnl_msg_packet_hdr nfqnl_msg_packet_hdr;
00210
00211 /*callback function type def's*/
00212
00219 typedef int (*frameworkfunc)(int, char **);
00220
00227 #ifndef __WIN32__
00228 typedef void    (*syssighandler)(int, siginfo_t *, void *);
00229 #else
00230 typedef void    (*syssighandler)(int, void*, void*);
```

```
00231 #endif
00232
00238 typedef void    (*threadcleanup)(void *);
00239
00245 typedef void    *(*threadfunc)(void *);
00246
00252 typedef int     (*threadsighandler)(int, void *);
00253
00259 typedef void    (*socketrecv)(struct fwsocket *, void *);
00260
00264 typedef void    (*objdestroy)(void *);
00265
00271 typedef int32_t (*blisthash)(const void *, int);
00272
00278 typedef void    (*blist_cb)(void *, void *);
00279
00285 typedef void    (*config_filecb)(struct bucket_list *, const char *, const char *);
00286
00291 typedef void    (*config_catcb)(struct bucket_list *, const char *);
00292
00297 typedef void    (*config_entrycb)(const char *, const char *);
00298
00300 typedef uint32_t (*nfqueue_cb)(struct nfq_data *, struct
      nfqnl_msg_packet_hdr *, char *, uint32_t, void *, uint32_t *, void **);
00301
00306 typedef void    (*radius_cb)(struct radius_packet *, void *);
00307
00310  enum framework_flags {
00312     FRAMEWORK_FLAG_DAEMON      = 1 << 0,
00314     FRAMEWORK_FLAG_NOGNU        = 1 << 1,
00319     FRAMEWORK_FLAG_DAEMONLOCK  = 1 << 2
00320 };
00321
00326 struct framework_core {
00328     const char *developer;
00330     const char *email;
00332     const char *www;
00334     const char *runfile;
00336     const char *progname;
00338     int  year;
00340     int  flock;
00342     struct sigaction *sa;
00345     syssighandler  sig_handler;
00348     int flags;
00349 };
00350
00351 void framework_mkcore(char *progname, char *name, char *email, char *web, int year, char *
      runfile, int flags, syssighandler sigfunc);
00352 extern int framework_init(int argc, char *argv[], frameworkfunc callback);
00353 void printgnu(const char *pname, int year, const char *dev, const char *email, const char *www);
00354 void daemonize();
00355 int lockpidfile(const char *runfile);
00356 extern struct thread_pvt *framework_mkthread(
      threadfunc, threadcleanup, threadsighandler, void *
      data, int flags);
00357 /* UNIX Socket*/
00358 extern struct fwsocket *unixsocket_server(const char *
      sock, int protocol, int mask, socketrecv read, void *data);
00359 extern struct fwsocket *unixsocket_client(const char *
      sock, int protocol, socketrecv read, void *data);
00360 /* Test if the thread is running when passed data from thread */
00361 extern int framework_threadok(void);
00362 extern int startthreads(void);
00363 extern void stopthreads(int join);
00364 int thread_signal(int sig);
00365
00366 /*
00367  * ref counted objects
00368  */
00369 extern int objlock(void *data);
00370 extern int objtrylock(void *data);
00371 extern int objunlock(void *data);
00372 extern int objcnt(void *data);
00373 extern int objsize(void *data);
00374 extern int objunref(void *data);
00375 extern int objref(void *data);
00376 extern void *objalloc(int size, objdestroy);
00377 void *objchar(const char *orig);
00378
00379 /*
00380  * hashed bucket lists
00381  */
00382 extern void *create_bucketlist(int bitmask, blisthash hash_function);
00383 extern int addtobucket(struct bucket_list *blist, void *data);
00384 extern void remove_bucket_item(struct bucket_list *blist, void *data);
00385 extern int bucket_list_cnt(struct bucket_list *blist);
00386 extern void *bucket_list_find_key(struct bucket_list *list, const void *key)
```

```
    ;
00387 extern void bucketlist_callback(struct bucket_list *blist,
    blist_cb callback, void *data2);
00388
00389 /*
00390  * iteration through buckets
00391  */
00392 extern struct bucket_loop *init_bucket_loop(struct
    bucket_list *blist);
00393 extern void *next_bucket_loop(struct bucket_loop *bloop);
00394 extern void remove_bucket_loop(struct bucket_loop *bloop);
00395
00396 /*include jenkins hash burttlebob*/
00397 extern uint32_t hashlittle(const void *key, size_t length, uint32_t initval);
00398
00399
00400 /*
00401  * Utilities RNG/MD5 used from the openssl library
00402  */
00403 extern void seedrand(void);
00404 extern int genrand(void *buf, int len);
00405 extern void sha512sum(unsigned char *buff, const void *data, unsigned long len);
00406 extern void sha256sum(unsigned char *buff, const void *data, unsigned long len);
00407 extern void sha1sum(unsigned char *buff, const void *data, unsigned long len);
00408 extern void md5sum(unsigned char *buff, const void *data, unsigned long len);
00409 extern void sha512sum2(unsigned char *buff, const void *data, unsigned long len, const void *
    data2, unsigned long len2);
00410 extern void sha256sum2(unsigned char *buff, const void *data, unsigned long len, const void *
    data2, unsigned long len2);
00411 extern void sha1sum2(unsigned char *buff, const void *data, unsigned long len, const void *data2,
    unsigned long len2);
00412 extern void md5sum2(unsigned char *buff, const void *data, unsigned long len, const void *data2,
    unsigned long len2);
00413 extern int sha512cmp(unsigned char *digest1, unsigned char *digest2);
00414 extern int sha256cmp(unsigned char *digest1, unsigned char *digest2);
00415 extern int sha1cmp(unsigned char *digest1, unsigned char *digest2);
00416 extern int md5cmp(unsigned char *digest1, unsigned char *digest2);
00417 extern void sha512hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
     unsigned long klen);
00418 extern void sha256hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
     unsigned long klen);
00419 extern void sha1hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
    unsigned long klen);
00420 extern void md5hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
    unsigned long klen);
00421 extern int strlenzero(const char *str);
00422 extern char *ltrim(char *str);
00423 extern char *rtrim(const char *str);
00424 extern char *trim(const char *str);
00425 extern uint64_t tvtontp64(struct timeval *tv);
00426 extern uint16_t checksum(const void *data, int len);
00427 extern uint16_t checksum_add(const uint16_t checksum, const void *data, int len);
00428 extern uint16_t verifysum(const void *data, int len, const uint16_t check);
00429 extern struct zobj *zcompress(uint8_t *buff, uint16_t len, uint8_t level);
00430 extern void zuncompress(struct zobj *buff, uint8_t *obuff);
00431 extern uint8_t *gzinflatebuf(uint8_t *buf_in, int buf_size, uint32_t *len);
00432 extern int is_gzip(uint8_t *buf, int buf_size);
00433 #ifdef __WIN32__
00434 extern void touch(const char *filename);
00435 #else
00436 extern void touch(const char *filename, uid_t user, gid_t group);
00437 #endif
00438 extern char *b64enc(const char *message, int nonl);
00439 extern char *b64enc_buf(const char *message, uint32_t len, int nonl);
00440
00441 /*IP Utilities*/
00442 extern struct fwsocket *make_socket(int family, int type, int
    proto, void *ssl);
00443 extern struct fwsocket *accept_socket(struct fwsocket *
    sock);
00444 extern struct fwsocket *sockconnect(int family, int stype, int
    proto, const char *ipaddr, const char *port, void *ssl);
00445 extern struct fwsocket *udpconnect(const char *ipaddr, const char *port, void *
    ssl);
00446 extern struct fwsocket *tcpconnect(const char *ipaddr, const char *port, void *
    ssl);
00447 extern struct fwsocket *sockbind(int family, int stype, int
    proto, const char *ipaddr, const char *port, void *ssl, int backlog);
00448 extern struct fwsocket *udpbind(const char *ipaddr, const char *port, void *
    ssl);
00449 extern struct fwsocket *tcpbind(const char *ipaddr, const char *port, void *
    ssl, int backlog);
00450 extern void close_socket(struct fwsocket *sock);
00451
00452 int score_ipv4(struct sockaddr_in *sa4, char *ipaddr, int iplen);
00453 int score_ipv6(struct sockaddr_in6 *sa6, char *ipaddr, int iplen);
00454
```

```
00455 #ifdef __WIN32
00456 const char *inet_ntop(int af, const void *src, char *dest, socklen_t size);
00457 struct ifinfo *get_ifinfo(const char *iface);
00458 #endif
00459
00460 int inet_lookup(int family, const char *host, void *addr, socklen_t len);
00461
00462 extern void socketclient(struct fwsocket *sock, void *data,
      socketrecv read, threadcleanup cleanup);
00463 extern void socketserver(struct fwsocket *sock, socketrecv connectfunc,
      socketrecv acceptfunc, threadcleanup cleanup, void *data);
00464 struct fwsocket *mcast_socket(const char *iface, int family, const char *mcastip, const
       char *port, int flags);
00465 const char *sockaddr2ip(union sockstruct *addr, char *buf, int len);
00466
00467 /*IP Utilities*/
00468 extern int checkipv6mask(const char *ipaddr, const char *network, uint8_t bits);
00469 extern void ipv4tcpchecksum(uint8_t *pkt);
00470 extern void ipv4udpchecksum(uint8_t *pkt);
00471 extern void ipv4icmpchecksum(uint8_t *pkt);
00472 extern void ipv4checksum(uint8_t *pkt);
00473 extern int packetchecksumv4(uint8_t *pkt);
00474 extern int packetchecksumv6(uint8_t *pkt);
00475 extern int packetchecksum(uint8_t *pkt);
00476 extern void rfc6296_map(struct natmap *map, struct in6_addr *ipaddr, int out);
00477 extern int rfc6296_map_add(char *intaddr, char *extaddr);
00478 const char *cidrtosn(int bitlen, char *buf, int size);
00479 const char *getnetaddr(const char *ipaddr, int cidr, char *buf, int size);
00480 const char *getbcaddr(const char *ipaddr, int cidr, char *buf, int size);
00481 const char *getfirstaddr(const char *ipaddr, int cidr, char *buf, int size);
00482 const char *getlastaddr(const char *ipaddr, int cidr, char *buf, int size);
00483 uint32_t cidrcnt(int bitlen);
00484 int reservedip(const char *ipaddr);
00485 char* ipv6to4prefix(const char *ipaddr);
00486 int check_ipv4(const char* ip, int cidr, const char *test);
00487 void mcast4_ip(struct in_addr *addr);
00488 void mcast6_ip(struct in6_addr *addr);
00489
00490 /*netfilter queue*/
00491 extern struct nfq_queue *nfqueue_attach(uint16_t pf, uint16_t
      num, uint8_t mode, uint32_t range, nfqueue_cb cb, void *data);
00492 extern uint16_t snprintf_pkt(struct nfq_data *tb, struct
      nfqnl_msg_packet_hdr *ph, uint8_t *pkt, char *buff, uint16_t len);
00493 extern struct nf_conntrack *nf_ctrack_buildct(uint8_t *pkt);
00494 extern uint8_t nf_ctrack_delete(uint8_t *pkt);
00495 extern uint8_t nf_ctrack_nat(uint8_t *pkt, uint32_t addr, uint16_t port, uint8_t dnat);
00496 extern void nf_ctrack_dump(void);
00497 extern struct nfct_struct *nf_ctrack_trace(void);
00498 extern void nf_ctrack_endtrace(struct nfct_struct *nfct);
00499 extern uint8_t nf_ctrack_init(void);
00500 extern void nf_ctrack_close(void);
00501
00502 /*interface functions*/
00503 extern int delete_kernvlan(char *ifname, int vid);
00504 extern int create_kernvlan(char *ifname, unsigned short vid);
00505 extern int delete_kernmac(char *macdev);
00506 extern int create_kernmac(char *ifname, char *macdev, unsigned char *mac);
00507 extern int interface_bind(char *iface, int protocol);
00508 extern void randhwaddr(unsigned char *addr);
00509 extern int create_tun(const char *ifname, const unsigned char *hwaddr, int flags);
00510 extern int ifrename(const char *oldname, const char *newname);
00511 extern int ifdown(const char *ifname, int flags);
00512 extern int ifup(const char *ifname, int flags);
00513 extern int ifhwaddr(const char *ifname, unsigned char *hwaddr);
00514 extern int set_interface_flags(int ifindex, int set, int clear);
00515 extern int get_iface_index(const char *ifname);
00516 extern int set_interface_addr(int ifindex, const unsigned char *hwaddr);
00517 extern int set_interface_name(int ifindex, const char *name);
00518 extern int set_interface_ipaddr(char *ifname, char *ipaddr);
00519 extern int get_ip6_addrprefix(const char *iface, unsigned char *prefix);
00520 extern void eui48to64(unsigned char *mac48, unsigned char *eui64);
00521 extern void closenetlink(void);
00522 extern int ifrename(const char *oldname, const char *newname);
00523 const char *get_ifipaddr(const char *iface, int family);
00524
00525 /*Radius utilities*/
00529 #define RAD_AUTH_HDR_LEN    20
00530
00532 #define RAD_AUTH_PACKET_LEN 4096
00533
00535 #define RAD_AUTH_TOKEN_LEN  16
00536
00538 #define RAD_MAX_PASS_LEN    128
00539
00541 #define RAD_ATTR_USER_NAME  1   /*string*/
00542
00544 #define RAD_ATTR_USER_PASSWORD  2   /*passwd*/
```

```
00545
00547 #define RAD_ATTR_NAS_IP_ADDR    4   /*ip*/
00548
00550 #define RAD_ATTR_NAS_PORT   5   /*int*/
00551
00553 #define RAD_ATTR_SERVICE_TYPE   6   /*int*/
00554
00556 #define RAD_ATTR_ACCTID     44
00557
00559 #define RAD_ATTR_PORT_TYPE  61  /*int*/
00560
00562 #define RAD_ATTR_EAP        79  /*oct*/
00563
00565 #define RAD_ATTR_MESSAGE    80  /*oct*/
00566
00568 enum RADIUS_CODE {
00570     RAD_CODE_AUTHREQUEST    =   1,
00572     RAD_CODE_AUTHACCEPT  =   2,
00574     RAD_CODE_AUTHREJECT  =   3,
00576     RAD_CODE_ACCTREQUEST    =   4,
00578     RAD_CODE_ACCTRESPONSE  =   5,
00580     RAD_CODE_AUTHCHALLENGE    =   11
00581 };
00584 extern void addradattrint(struct radius_packet *packet, char type, unsigned int
      val);
00585 extern void addradattrip(struct radius_packet *packet, char type, char *ipaddr);
00586 extern void addradattrstr(struct radius_packet *packet, char type, char *str);
00587 extern struct radius_packet *new_radpacket(unsigned char
      code);
00588 extern int send_radpacket(struct radius_packet *packet, const char *userpass,
      radius_cb read_cb, void *cb_data);
00589 extern void add_radserver(const char *ipaddr, const char *auth, const char *acct, const char *
      secret, int timeout);
00590 extern unsigned char *radius_attr_first(struct radius_packet *packet);
00591 extern unsigned char *radius_attr_next(struct radius_packet *packet, unsigned
      char *attr);
00592
00593 /*SSL Socket utilities*/
00594 extern void sslstartup(void);
00595 extern void *tlsv1_init(const char *cacert, const char *cert, const char *key, int verify);
00596 extern void *sslv2_init(const char *cacert, const char *cert, const char *key, int verify);
00597 extern void *sslv3_init(const char *cacert, const char *cert, const char *key, int verify);
00598 extern void *dtlsv1_init(const char *cacert, const char *cert, const char *key, int verify);
00599
00600 extern int socketread(struct fwsocket *sock, void *buf, int num);
00601 extern int socketwrite(struct fwsocket *sock, const void *buf, int num);
00602 /*the following are only needed on server side of a dgram connection*/
00603 extern int socketread_d(struct fwsocket *sock, void *buf, int num, union
      sockstruct *addr);
00604 extern int socketwrite_d(struct fwsocket *sock, const void *buf, int num, union
      sockstruct *addr);
00605
00606 extern void ssl_shutdown(void *ssl, int sock);
00607 extern void tlsaccept(struct fwsocket *sock, struct ssldata *orig);
00608 extern struct fwsocket *dtls_listenssl(struct fwsocket *
      sock);
00609 extern void startsslclient(struct fwsocket *sock);
00610
00611 /*config file parsing functions*/
00612 extern void unrefconfigfiles(void);
00613 extern int process_config(const char *configname, const char *configfile);
00614 extern struct bucket_loop *get_category_loop(const char *configname);
00615 extern struct bucket_list *get_category_next(struct
      bucket_loop *cloop, char *name, int len);
00616 extern struct bucket_list *get_config_category(const char *configname, const
      char *category);
00617 extern struct config_entry *get_config_entry(struct
      bucket_list *categories, const char *item);
00618 extern void config_file_callback(config_filecb file_cb);
00619 extern void config_cat_callback(struct bucket_list *categories,
      config_catcb entry_cb);
00620 extern void config_entry_callback(struct bucket_list *entries,
      config_entrycb entry_cb);
00621
00622 /*Forward Decl*/
00625 typedef struct xml_node xml_node;
00628 typedef struct xml_search xml_search;
00631 typedef struct xml_doc xml_doc;
00634 typedef struct xslt_doc xslt_doc;
00635
00636 /*XML*/
00639 struct xml_attr {
00641     const char  *name;
00643     const char  *value;
00644 };
00645
00648 struct xml_node {
```

```
00650     const char      *name;
00652     const char      *value;
00654     const char      *key;
00656     struct bucket_list   *attrs;
00658     void            *nodeptr;
00659 };
00660
00661 extern struct xml_doc *xml_loaddoc(const char *docfile, int validate);
00662 extern struct xml_doc *xml_loadbuf(const uint8_t *buffer, uint32_t len, int validate);
00663 extern struct xml_node *xml_getfirstnode(struct
      xml_search *xpsearch, void **iter);
00664 extern struct xml_node *xml_getnextnode(void *iter);
00665 extern struct bucket_list *xml_getnodes(struct xml_search *xpsearch);
00666 extern struct xml_search *xml_xpath(struct xml_doc *xmldata, const char *xpath,
      const char *attrkey);
00667 extern int xml_nodecount(struct xml_search *xsearch);
00668 extern struct xml_node *xml_getnode(struct xml_search *xsearch, const char *
      key);
00669 extern const char *xml_getattr(struct xml_node *xnode, const char *attr);
00670 extern void xml_modify(struct xml_doc *xmldoc, struct xml_node *xnode, const char
      *value);
00671 extern void xml_setattr(struct xml_doc *xmldoc, struct
      xml_node *xnode, const char *name, const char *value);
00672 extern struct xml_node *xml_addnode(struct xml_doc *xmldoc, const char *xpath,
      const char *name, const char *value, const char *attrkey, const char *keyval);
00673 void xml_appendnode(struct xml_doc *xmldoc, const char *xpath, struct
      xml_node *child);
00674 void xml_unlink(struct xml_node *xnode);
00675 extern void xml_delete(struct xml_node *xnode);
00676 extern char *xml_getbuffer(void *buffer);
00677 extern void *xml_doctobuffer(struct xml_doc *xmldoc);
00678 extern const char *xml_getrootname(struct xml_doc *xmldoc);
00679 extern struct xml_node *xml_getrootnode(struct xml_doc *xmldoc);
00680 extern void xml_savefile(struct xml_doc *xmldoc, const char *file, int format, int
      compress);
00681 extern void xml_createpath(struct xml_doc *xmldoc, const char *xpath);
00682 extern void xml_init();
00683 extern void xml_close();
00684
00685 /*XSLT*/
00686 struct xslt_doc *xslt_open(const char *xsltfile);
00687 void xslt_addparam(struct xslt_doc *xsltdoc, const char *param, const char *value);
00688 void xslt_apply(struct xml_doc *xmldoc, struct xslt_doc *xsltdoc, const char *
      filename, int comp);
00689 void *xslt_apply_buffer(struct xml_doc *xmldoc, struct
      xslt_doc *xsltdoc);
00690 void xslt_init();
00691 void xslt_close();
00692
00693 /* LDAP */
00697 enum ldap_starttls {
00699     LDAP_STARTTLS_NONE,
00701     LDAP_STARTTLS_ATTEMPT,
00703     LDAP_STARTTLS_ENFORCE
00704 };
00705
00707 enum ldap_attrtype {
00709     LDAP_ATTRTYPE_CHAR,
00711     LDAP_ATTRTYPE_B64,
00713     LDAP_ATTRTYPE_OCTET
00714 };
00715
00717 struct ldap_rdn {
00719     const char *name;
00721     const char *value;
00723     struct ldap_rdn *next;
00725     struct ldap_rdn *prev;
00726 };
00727
00729 struct ldap_attrval {
00731     int len;
00733     enum ldap_attrtype type;
00735     char *buffer;
00736 };
00737
00739 struct ldap_attr {
00741     const char *name;
00743     int count;
00745     struct ldap_attrval **vals;
00747     struct ldap_attr *next;
00749     struct ldap_attr *prev;
00750 };
00751
00753 struct ldap_entry {
00755     const char *dn;
00757     const char *dnufn;
00759     int rdncnt;
```

```
00761     struct ldap_rdn **rdn;
00763     struct ldap_attr *list;
00765     struct bucket_list *attrs;
00767     struct ldap_attr *first_attr;
00769     struct ldap_entry *next;
00771     struct ldap_entry *prev;
00772 };
00773
00775 struct ldap_results {
00777     int count;
00779     struct ldap_entry *first_entry;
00781     struct bucket_list *entries;
00782 };
00783
00785 typedef struct ldap_conn ldap_conn;
00787 typedef struct ldap_modify ldap_modify;
00789 typedef struct ldap_add ldap_add;
00792 extern struct ldap_conn *ldap_connect(const char *uri, enum
      ldap_starttls starttls,int timelimit, int limit, int debug, int *err);
00793 extern int ldap_simplebind(struct ldap_conn *ld, const char *dn, const char *passwd
      );
00794 extern int ldap_saslbind(struct ldap_conn *ld, const char *mech, const char *realm,
      const char *authcid,
00795                          const char *passwd, const char *authzid);
00796 extern int ldap_simplerebind(struct ldap_conn *ld, const char *initialdn, const
      char *initialpw, const char *base, const char *filter,
00797                              const char *uidrdn, const char *uid, const char *passwd);
00798 extern const char *ldap_errmsg(int res);
00799
00800 extern struct ldap_results *ldap_search_sub(struct
      ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res, ...);
00801 extern struct ldap_results *ldap_search_one(struct
      ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res, ...);
00802 extern struct ldap_results *ldap_search_base(struct
      ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res, ...);
00803
00804 extern void ldap_unref_entry(struct ldap_results *results, struct
      ldap_entry *entry);
00805 extern void ldap_unref_attr(struct ldap_entry *entry, struct
      ldap_attr *attr);
00806 extern struct ldap_entry *ldap_getentry(struct
      ldap_results *results, const char *dn);
00807 extern struct ldap_attr *ldap_getattr(struct ldap_entry *entry, const char *
      attr);
00808
00809 extern struct ldap_modify *ldap_modifyinit(const char *
      dn);
00810 extern int ldap_mod_del(struct ldap_modify *lmod, const char *attr, ...);
00811 extern int ldap_mod_add(struct ldap_modify *lmod, const char *attr, ...);
00812 extern int ldap_mod_rep(struct ldap_modify *lmod, const char *attr, ...);
00813 extern int ldap_domodify(struct ldap_conn *ld, struct
      ldap_modify *lmod);
00814
00815 extern int ldap_mod_rematttr(struct ldap_conn *ldap, const char *
      dn, const char *attr);
00816 extern int ldap_mod_delattr(struct ldap_conn *ldap, const char *
      dn, const char *attr, const char *value);
00817 extern int ldap_mod_addattr(struct ldap_conn *ldap, const char *
      dn, const char *attr, const char *value);
00818 extern int ldap_mod_repattr(struct ldap_conn *ldap, const char *
      dn, const char *attr, const char *value);
00819
00824 struct basic_auth {
00826     const char *user;
00828     const char *passwd;
00829 };
00830
00832 struct curlbuf {
00834     uint8_t *header;
00836     uint8_t *body;
00838     char *c_type;
00840     size_t hsize;
00842     size_t bsize;
00843 };
00844
00846 typedef struct curl_post curl_post;
00847
00853 typedef struct basic_auth *(*curl_authcb)(const char*, const char*, void*);
00854
00862 typedef int (*curl_progress_func)(void*, double, double, double, double);
00863
00867 typedef void(*curl_progress_pause)(void*, int);
00868
00876 typedef void *(*curl_progress_newdata)(void*);
00877
00880 int curlinit(void);
00881 void curlclose(void);
```

```
00882 struct basic_auth *curl_newauth(const char *user, const char *
      passwd);
00883 struct curlbuf *curl_geturl(const char *def_url, struct
      basic_auth *bauth, curl_authcb authcb,void *data);
00884 void curl_setprogress(curl_progress_func cb,
      curl_progress_pause p_cb, curl_progress_newdata d_cb, void *data);
00885 void curl_setauth_cb(curl_authcb auth_cb, void *data);
00886 struct curl_post *curl_newpost(void);
00887 void curl_postitem(struct curl_post *post, const char *name, const char *item);
00888 struct curlbuf *curl_posturl(const char *def_url, struct
      basic_auth *bauth, struct curl_post *post, curl_authcb authcb,void *data);
00889 struct curlbuf *curl_ungzip(struct curlbuf *cbuf);
00890 extern struct xml_doc *curl_buf2xml(struct curlbuf *cbuf);
00891 char *url_escape(char *url);
00892 char *url_unescape(char *url);
00893
00894
00895 /*File Utils*/
00896 int is_file(const char *path);
00897 int is_dir(const char *path);
00898 int is_exec(const char *path);
00899 #ifdef __WIN32__
00900 int mk_dir(const char *dir);
00901 #else
00902 int mk_dir(const char *dir, mode_t mode, uid_t user, gid_t group);
00903 #endif
00904
00909 #define JHASH_INITVAL           0xdeadbeef
00910
00914 #define jenhash(key, length, initval)   hashlittle(key, length, (initval) ? initval : JHASH_INITVAL);
00915
00918 #define clearflag(obj, flag) \
00919 objlock(obj);\
00920 obj->flags &= ~flag;\
00921 objunlock(obj)
00922
00925 #define setflag(obj, flag) \
00926 objlock(obj);\
00927 obj->flags |= flag; \
00928 objunlock(obj)
00929
00932 #define testflag(obj, flag) \
00933 (objlock(obj) | (obj->flags & flag) | objunlock(obj))
00934
00949 #define FRAMEWORK_MAIN(progname, name, email, www, year, runfile, flags, sighfunc) \
00950 static int  framework_main(int argc, char *argv[]); \
00951 int  main(int argc, char *argv[]) { \
00952     framework_mkcore(progname, name, email, www, year, runfile, flags, sighfunc); \
00953     return (framework_init(argc, argv, framework_main)); \
00954 } \
00955 static int  framework_main(int argc, char *argv[])
00956
00959 #define ALLOC_CONST(const_var, val) { \
00960         char *tmp_char; \
00961         if (val) { \
00962             tmp_char = (char*)malloc(strlen(val) + 1); \
00963             strcpy(tmp_char, val); \
00964             const_var = (const char*)tmp_char; \
00965         } else { \
00966             const_var = NULL; \
00967         } \
00968     }
00969
00976 #define DTS_OJBREF_CLASS(classtype) \
00977 void *operator new(size_t sz) {\
00978     return objalloc(sz, &classtype::dts_unref_classtype);\
00979 }\
00980 void operator delete(void *obj) {\
00981 }\
00982 static void dts_unref_classtype(void *data) {\
00983     delete (classtype*)data;\
00984 }\
00985 ~classtype()
00986
00987 #ifdef __cplusplus
00988 }
00989 #endif
00990 #endif
```

14.22 src/interface.c File Reference

Wrapper arround Linux libnetlink for managing network interfaces.

```
#include <netinet/in.h>
#include <linux/if_vlan.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>
#include <linux/if_tun.h>
#include <linux/if_arp.h>
#include <linux/sockios.h>
#include <linux/if.h>
#include <ifaddrs.h>
#include <sys/ioctl.h>
#include <netdb.h>
#include <sys/time.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include "include/dtsapp.h"
#include "libnetlink/include/libnetlink.h"
#include "libnetlink/include/ll_map.h"
#include "libnetlink/include/utils.h"
```

Data Structures

- struct iplink_req

  IP Netlink request.
- struct ipaddr_req

  IP Netlink IP addr request.

Enumerations

- enum ipv4_score { IPV4_SCORE_ZEROCONF = 1 << 0, IPV4_SCORE_RESERVED = 1 << 1, IPV4_SCORE_ROUTABLE = 1 << 2 }

  Order of precidence of ipv4.
- enum ipv6_score { IPV6_SCORE_RESERVED = 1 << 0, IPV6_SCORE_SIXIN4 = 1 << 1, IPV6_SCORE_ROUTABLE = 1 << 2 }

  Return best ipv6 address in order of FFC/7 2002/16 ...

Functions

- void closenetlink ()

  Close netlink socket on application termination.
- int get_iface_index (const char *ifname)

  Get the netlink interface for a named interface.
- int delete_kernvlan (char *ifname, int vid)

  Delete a VLAN.
- int create_kernvlan (char *ifname, unsigned short vid)

  Create a VLAN on a interface.
- int delete_kernmac (char *ifname)

  Delete Kernel MAC VLAN.
- int create_kernmac (char *ifname, char *macdev, unsigned char *mac)

  Create a kernal MAC VLAN.

- int set_interface_flags (int ifindex, int set, int clear)

  Alter interface flags.
- int set_interface_addr (int ifindex, const unsigned char *hwaddr)

  Set interface MAC addr.
- int set_interface_name (int ifindex, const char *name)

  Rename interface.
- int interface_bind (char *iface, int protocol)

  Bind to device fd may be a existing socket.
- void randhwaddr (unsigned char *addr)

  create random MAC address
- int create_tun (const char *ifname, const unsigned char *hwaddr, int flags)

  Create a tunnel device.
- int ifdown (const char *ifname, int flags)

  Set interface down.
- int ifup (const char *ifname, int flags)

  Set interface up.
- int ifrename (const char *oldname, const char *newname)

  Rename interface helper.
- int ifhwaddr (const char *ifname, unsigned char *hwaddr)

  Get MAC addr for interface.
- int set_interface_ipaddr (char *ifname, char *ipaddr)

  Set IP addr on interface.
- void eui48to64 (unsigned char *mac48, unsigned char *eui64)

  Generate IPv6 address from mac address.
- int get_ip6_addrprefix (const char *iface, unsigned char *prefix)

  Generate Unique Local IPv6 Unicast Addresses RFC 4193.
- int score_ipv4 (struct sockaddr_in *sa4, char *ipaddr, int iplen)

  Return a score for a IPv4 addrress.
- int score_ipv6 (struct sockaddr_in6 *sa6, char *ipaddr, int iplen)

  Return a score for a IPv6 addrress.
- const char * get_ifipaddr (const char *iface, int family)

  Find best IP adress for a interface.

14.22.1 Detailed Description

Wrapper arround Linux libnetlink for managing network interfaces.

Definition in file interface.c.

14.23 interface.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

```
00017 */
00018
00026 #ifndef __WIN32
00027 #include <netinet/in.h>
00028 #include <linux/if_vlan.h>
00029 #include <linux/if_ether.h>
00030 #include <linux/if_packet.h>
00031 #include <linux/if_tun.h>
00032 #include <linux/if_arp.h>
00033 #include <linux/sockios.h>
00034 #include <linux/if.h>
00035 #include <ifaddrs.h>
00036 #include <sys/ioctl.h>
00037 #include <netdb.h>
00038 #else
00039 #include <winsock2.h>
00040 #include <ws2tcpip.h>
00041 #define ETH_ALEN 8
00042 #endif
00043
00044 #include <sys/time.h>
00045 #include <fcntl.h>
00046 #include <stdio.h>
00047 #include <stdint.h>
00048 #include <string.h>
00049 #include <unistd.h>
00050
00051 #include "include/dtsapp.h"
00052 #ifndef __WIN32
00053 #include "libnetlink/include/libnetlink.h"
00054 #include "libnetlink/include/ll_map.h"
00055 #include "libnetlink/include/utils.h"
00056
00057 static struct rtnl_handle *nlh;
00058
00059 #endif
00060
00061
00063 enum ipv4_score {
00065     IPV4_SCORE_ZEROCONF = 1 << 0,
00067     IPV4_SCORE_RESERVED = 1 << 1,
00069     IPV4_SCORE_ROUTABLE = 1 << 2
00070 };
00071
00073 enum ipv6_score {
00075     IPV6_SCORE_RESERVED = 1 << 0,
00077     IPV6_SCORE_SIXIN4 = 1 << 1,
00079     IPV6_SCORE_ROUTABLE = 1 << 2
00080 };
00081
00082 #ifndef __WIN32
00083
00085 struct iplink_req {
00087     struct nlmsghdr     n;
00089     struct ifinfomsg    i;
00091     char            buf[1024];
00092 };
00093
00095 struct ipaddr_req {
00097     struct nlmsghdr     n;
00099     struct ifaddrmsg    i;
00101     char            buf[1024];
00102 };
00103
00104 static void nlhandle_free(void *data) {
00105     struct rtnl_handle *nlh = data;
00106
00107     if (data) {
00108         rtnl_close(nlh);
00109     }
00110 }
00111
00112 static struct rtnl_handle *nlhandle(int subscriptions) {
00113     struct rtnl_handle *nlh;
00114
00115     if (!(nlh = objalloc(sizeof(*nlh), nlhandle_free)) || (rtnl_open(nlh, 0))) {
00116         if (nlh) {
00117             objunref(nlh);
00118         }
00119         return (NULL);
00120     }
00121
00122     /*initilise the map*/
00123     ll_init_map(nlh, 0);
00124     objref(nlh);
00125
00126     return (nlh);
```

```
00127 }
00128
00130 extern void closenetlink() {
00131     if (nlh) {
00132         objunref(nlh);
00133     }
00134 }
00135
00139 extern int get_iface_index(const char *ifname) {
00140     int ifindex;
00141
00142     if (!objref(nlh) && !(nlh = nlhandle(0))) {
00143         return (0);
00144     }
00145
00146     objlock(nlh);
00147     ll_init_map(nlh, 1);
00148     objunlock(nlh);
00149
00150     ifindex = ll_name_to_index(ifname);
00151
00152     objunref(nlh);
00153     return (ifindex);
00154 }
00155
00159 static int delete_interface(char *iface) {
00160     struct iplink_req *req;
00161     int ifindex, ret;
00162
00163     /*check ifname grab a ref to nlh or open it*/
00164     if (strlenzero(iface) || (strlen(iface) > IFNAMSIZ) ||
00165             (!objref(nlh) && !(nlh = nlhandle(0)))) {
00166         return (-1);
00167     }
00168
00169     /*set the index of base interface*/
00170     if (!(ifindex = get_iface_index(iface))) {
00171         objunref(nlh);
00172         return (-1);
00173     }
00174
00175     if (!(req = objalloc(sizeof(*req), NULL))) {
00176         objunref(nlh);
00177         return (-1);
00178     }
00179
00180     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00181     req->n.nlmsg_type = RTM_DELLINK;
00182     req->n.nlmsg_flags = NLM_F_REQUEST;
00183
00184     /*config base/dev/mac*/
00185     req->i.ifi_index = ifindex;
00186
00187     objlock(nlh);
00188     ret = rtnl_talk(nlh, &req->n, 0, 0, NULL);
00189     objunlock(nlh);
00190
00191     objunref(nlh);
00192     objunref(req);
00193
00194     return (ret);
00195 }
00196
00201 extern int delete_kernvlan(char *ifname, int vid) {
00202     char iface[IFNAMSIZ+1];
00203
00204     /*check ifname grab a ref to nlh or open it*/
00205     snprintf(iface, IFNAMSIZ, "%s.%i", ifname, vid);
00206     return (delete_interface(iface));
00207 }
00208
00209
00214 extern int create_kernvlan(char *ifname, unsigned short vid) {
00215     struct iplink_req *req;
00216     char iface[IFNAMSIZ+1];
00217     struct rtattr *data, *linkinfo;
00218     char *type = "vlan";
00219     int ifindex, ret;
00220
00221     if (strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
00222             (!objref(nlh) && !(nlh = nlhandle(0)))) {
00223         return (-1);
00224     }
00225
00226     /*set the index of base interface*/
00227     if (!(ifindex = get_iface_index(ifname))) {
00228         objunref(nlh);
```

```
00229         return (-1);
00230     }
00231
00232     if (!(req = objalloc(sizeof(*req), NULL))) {
00233         objunref(nlh);
00234         return (-1);
00235     }
00236
00237     snprintf(iface, IFNAMSIZ, "%s.%i", ifname, vid);
00238     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00239     req->n.nlmsg_type = RTM_NEWLINK;
00240     req->n.nlmsg_flags = NLM_F_CREATE | NLM_F_EXCL | NLM_F_REQUEST;
00241
00242     /*config base/dev/mac*/
00243     addattr_l(&req->n, sizeof(*req), IFLA_LINK, &ifindex, sizeof(ifindex));
00244     addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, iface, strlen(iface));
00245
00246     /*type*/
00247     linkinfo  = NLMSG_TAIL(&req->n);
00248     addattr_l(&req->n, sizeof(*req), IFLA_LINKINFO, NULL, 0);
00249     addattr_l(&req->n, sizeof(*req), IFLA_INFO_KIND, type, strlen(type));
00250
00251     /*vid*/
00252     data = NLMSG_TAIL(&req->n);
00253     addattr_l(&req->n, sizeof(*req), IFLA_INFO_DATA, NULL, 0);
00254     addattr_l(&req->n, sizeof(*req), IFLA_VLAN_ID, &vid, sizeof(vid));
00255
00256     data->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)data;
00257     linkinfo->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)linkinfo;
00258
00259     objlock(nlh);
00260     ret = rtnl_talk(nlh, &req->n, 0, 0, NULL);
00261     objunlock(nlh);
00262
00263     objunref(nlh);
00264     objunref(req);
00265
00266     return (ret);
00267 }
00268
00272 extern int delete_kernmac(char *ifname) {
00273
00274     return (delete_interface(ifname));
00275 }
00276
00282 extern int create_kernmac(char *ifname, char *macdev, unsigned char *mac) {
00283     struct iplink_req *req;
00284     struct rtattr *data, *linkinfo;
00285     unsigned char lmac[ETH_ALEN];
00286     char *type = "macvlan";
00287     int ifindex, ret;
00288
00289     if (strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
00290             strlenzero(macdev) || (strlen(macdev) > IFNAMSIZ) ||
00291             (!objref(nlh) && !(nlh = nlhandle(0)))) {
00292         return (-1);
00293     }
00294
00295     /*set the index of base interface*/
00296     if (!(ifindex = get_iface_index(ifname))) {
00297         objunref(nlh);
00298         return (-1);
00299     }
00300
00301     if (!mac) {
00302         randhwaddr(lmac);
00303     } else {
00304         strncpy((char *)lmac, (char *)mac, ETH_ALEN);
00305     }
00306
00307     if (!(req = objalloc(sizeof(*req), NULL))) {
00308         objunref(nlh);
00309         return (-1);
00310     }
00311
00312     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00313     req->n.nlmsg_type = RTM_NEWLINK;
00314     req->n.nlmsg_flags = NLM_F_CREATE | NLM_F_EXCL | NLM_F_REQUEST;
00315
00316     /*config base/dev/mac*/
00317     addattr_l(&req->n, sizeof(*req), IFLA_LINK, &ifindex, 4);
00318     addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, macdev, strlen(macdev));
00319     addattr_l(&req->n, sizeof(*req), IFLA_ADDRESS, lmac, ETH_ALEN);
00320
00321     /*type*/
00322     linkinfo  = NLMSG_TAIL(&req->n);
00323     addattr_l(&req->n, sizeof(*req), IFLA_LINKINFO, NULL, 0);
```

```
00324     addattr_l(&req->n, sizeof(*req), IFLA_INFO_KIND, type, strlen(type));
00325
00326     /*mode*/
00327     data = NLMSG_TAIL(&req->n);
00328     addattr_l(&req->n, sizeof(*req), IFLA_INFO_DATA, NULL, 0);
00329     addattr32(&req->n, sizeof(*req), IFLA_MACVLAN_MODE, MACVLAN_MODE_PRIVATE);
00330     data->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)data;
00331     linkinfo->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)linkinfo;
00332
00333     objlock(nlh);
00334     ret = rtnl_talk(nlh, &req->n, 0, 0, NULL);
00335     objunlock(nlh);
00336
00337     objunref(nlh);
00338     objunref(req);
00339
00340     return (ret);
00341 }
00342
00348 extern int set_interface_flags(int ifindex, int set, int clear) {
00349     struct iplink_req *req;
00350     int flags;
00351
00352     if (!objref(nlh) && !(nlh = nlhandle(0))) {
00353         return (-1);
00354     }
00355
00356     flags = ll_index_to_flags(ifindex);
00357
00358     flags |= set;
00359     flags &= ~(clear);
00360
00361     if (!(req = objalloc(sizeof(*req), NULL))) {
00362         objunref(nlh);
00363         return (-1);
00364     }
00365
00366     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00367     req->n.nlmsg_type = RTM_NEWLINK;
00368     req->n.nlmsg_flags = NLM_F_REQUEST;
00369
00370     /*config base/dev/mac*/
00371     req->i.ifi_index = ifindex;
00372     req->i.ifi_flags = flags;
00373     req->i.ifi_change = set | clear;
00374
00375     objlock(nlh);
00376     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00377     objunlock(nlh);
00378
00379     objunref(nlh);
00380     objunref(req);
00381     return (0);
00382 }
00383
00388 extern int set_interface_addr(int ifindex, const unsigned char *hwaddr) {
00389     struct iplink_req *req;
00390
00391     if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
00392         return (-1);
00393     }
00394
00395     if (!(req = objalloc(sizeof(*req), NULL))) {
00396         objunref(nlh);
00397         return (-1);
00398     }
00399
00400     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00401     req->n.nlmsg_type = RTM_NEWLINK;
00402     req->n.nlmsg_flags = NLM_F_REQUEST;
00403     req->i.ifi_index = ifindex;
00404
00405     /*config base/dev/mac*/
00406     addattr_l(&req->n, sizeof(*req), IFLA_ADDRESS, hwaddr, ETH_ALEN);
00407
00408     objlock(nlh);
00409     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00410     objunlock(nlh);
00411
00412     objunref(nlh);
00413     objunref(req);
00414     return (0);
00415 }
00416
00421 extern int set_interface_name(int ifindex, const char *name) {
00422     struct iplink_req *req;
00423
```

```
00424     if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
00425         return (-1);
00426     }
00427
00428     if (!(req = objalloc(sizeof(*req), NULL))) {
00429         objunref(nlh);
00430         return (-1);
00431     }
00432
00433     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00434     req->n.nlmsg_type = RTM_NEWLINK;
00435     req->n.nlmsg_flags = NLM_F_REQUEST;
00436     req->i.ifi_index = ifindex;
00437
00438     addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, name, strlen((char *)name));
00439
00440     objlock(nlh);
00441     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00442     objunlock(nlh);
00443
00444     objunref(nlh);
00445     objunref(req);
00446     return (0);
00447 }
00448
00453 extern int interface_bind(char *iface, int protocol) {
00454     struct sockaddr_ll sll;
00455     int proto = htons(protocol);
00456     int fd, ifindex;
00457
00458     /*set the network dev up*/
00459     if (!(ifindex = get_iface_index(iface))) {
00460         return (-1);
00461     }
00462     set_interface_flags(ifindex, IFF_UP | IFF_RUNNING, 0);
00463
00464     /* open network raw socket */
00465     if ((fd = socket(PF_PACKET, SOCK_RAW,  proto)) < 0) {
00466         return (-1);
00467     }
00468
00469     /*bind to the interface*/
00470     memset(&sll, 0, sizeof(sll));
00471     sll.sll_family = PF_PACKET;
00472     sll.sll_protocol = proto;
00473     sll.sll_ifindex = ifindex;
00474     if (bind(fd, (struct sockaddr *)&sll, sizeof(sll)) < 0) {
00475         perror("bind failed");
00476         close(fd);
00477         return (-1);
00478     }
00479
00480     return (fd);
00481 }
00482
00485 extern void randhwaddr(unsigned char *addr) {
00486     genrand(addr, ETH_ALEN);
00487     addr [0] &= 0xfe;       /* clear multicast bit */
00488     addr [0] |= 0x02;       /* set local assignment bit (IEEE802) */
00489 }
00490
00496 extern int create_tun(const char *ifname, const unsigned char *hwaddr, int flags) {
00497     struct ifreq ifr;
00498     int fd, ifindex;
00499     char *tundev = "/dev/net/tun";
00500
00501     /* open the tun/tap clone dev*/
00502     if ((fd = open(tundev, O_RDWR)) < 0) {
00503         return (-1);
00504     }
00505
00506     /* configure the device*/
00507     memset(&ifr, 0, sizeof(ifr));
00508     ifr.ifr_flags = flags;
00509     strncpy(ifr.ifr_name, ifname, IFNAMSIZ);
00510     if (ioctl(fd, TUNSETIFF, (void *)&ifr) < 0 ) {
00511         perror("ioctl(TUNSETIFF) failed\n");
00512         close(fd);
00513         return (-1);
00514     }
00515
00516     if (!(ifindex = get_iface_index(ifname))) {
00517         return (-1);
00518     }
00519
00520     /* set the MAC address*/
00521     if (hwaddr) {
```

```
00522         set_interface_addr(ifindex, hwaddr);
00523     }
00524
00525     /*set the network dev up*/
00526     set_interface_flags(ifindex, IFF_UP | IFF_RUNNING | IFF_MULTICAST | IFF_BROADCAST, 0
      );
00527
00528     return (fd);
00529 }
00530
00535 extern int ifdown(const char *ifname, int flags) {
00536     int ifindex;
00537
00538     /*down the device*/
00539     if (!(ifindex = get_iface_index(ifname))) {
00540         return (-1);
00541     }
00542
00543     /*set the network dev up*/
00544     set_interface_flags(ifindex, 0, IFF_UP | IFF_RUNNING | flags);
00545
00546     return (0);
00547 }
00548
00553 extern int ifup(const char *ifname, int flags) {
00554     int ifindex;
00555
00556     /*down the device*/
00557     if (!(ifindex = get_iface_index(ifname))) {
00558         return (-1);
00559     }
00560
00561     /*set the network dev up*/
00562     set_interface_flags(ifindex, IFF_UP | IFF_RUNNING | flags, 0);
00563
00564     return (0);
00565 }
00566
00571 extern int ifrename(const char *oldname, const char *newname) {
00572     int ifindex;
00573
00574     ifdown(oldname, 0);
00575
00576     if (!(ifindex = get_iface_index(oldname))) {
00577         return (-1);
00578     }
00579     set_interface_name(ifindex, newname);
00580
00581     return (0);
00582 }
00583
00588 extern int ifhwaddr(const char *ifname, unsigned char *hwaddr) {
00589     int ifindex;
00590
00591     if (!hwaddr || strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
00592             (!objref(nlh) && !(nlh = nlhandle(0)))) {
00593         return (-1);
00594     }
00595
00596     /*set the index of base interface*/
00597     if (!(ifindex = get_iface_index(ifname))) {
00598         objunref(nlh);
00599         return (-1);
00600     }
00601
00602     ll_index_to_addr(ifindex, hwaddr, ETH_ALEN);
00603     objunref(nlh);
00604     return (0);
00605 }
00606
00611 extern int set_interface_ipaddr(char *ifname, char *ipaddr) {
00612     struct ipaddr_req *req;
00613     inet_prefix lcl;
00614     int ifindex, bcast;
00615
00616     if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
00617         return (-1);
00618     }
00619
00620     if (!(req = objalloc(sizeof(*req), NULL))) {
00621         objunref(nlh);
00622         return (-1);
00623     }
00624
00625     /*set the index of base interface*/
00626     if (!(ifindex = get_iface_index(ifname))) {
00627         objunref(nlh);
```

```
00628         return (-1);
00629     }
00630
00631     req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifaddrmsg));
00632     req->n.nlmsg_type = RTM_NEWADDR;
00633     req->n.nlmsg_flags = NLM_F_REQUEST | NLM_F_EXCL | NLM_F_CREATE;
00634
00635     req->i.ifa_scope = RT_SCOPE_HOST;
00636     req->i.ifa_index = ifindex;
00637
00638     get_prefix(&lcl, ipaddr, AF_UNSPEC);
00639     req->i.ifa_family = lcl.family;
00640     req->i.ifa_prefixlen = lcl.bitlen;
00641
00642     addattr_l(&req->n, sizeof(*req), IFA_LOCAL, &lcl.data, lcl.bytelen);
00643     addattr_l(&req->n, sizeof(*req), IFA_ADDRESS, &lcl.data, lcl.bytelen);
00644     if (lcl.family == AF_INET) {
00645         bcast = htonl((1 << (32 - lcl.bitlen)) - 1);
00646         addattr32(&req->n, sizeof(*req), IFA_BROADCAST, lcl.data[0] | bcast);
00647     }
00648
00649     objlock(nlh);
00650     rtnl_talk(nlh, &req->n, 0, 0, NULL);
00651     objunlock(nlh);
00652
00653     objunref(nlh);
00654     objunref(req);
00655     return (0);
00656 }
00657 #endif
00658
00668 extern void eui48to64(unsigned char *mac48, unsigned char *eui64) {
00669     eui64[0] = (mac48[0] & 0xFE) ^ 0x02; /*clear multicast bit and flip local asignment*/
00670     eui64[1] = mac48[1];
00671     eui64[2] = mac48[2];
00672     eui64[3] = 0xFF;
00673     eui64[4] = 0xFE;
00674     eui64[5] = mac48[3];
00675     eui64[6] = mac48[4];
00676     eui64[7] = mac48[5];
00677 }
00678
00686 #ifndef __WIN32
00687 extern int get_ip6_addrprefix(const char *iface, unsigned char *prefix) {
00688     uint64_t ntpts;
00689     unsigned char eui64[8];
00690     unsigned char sha1[20];
00691     unsigned char mac48[ETH_ALEN];
00692     struct timeval tv;
00693
00694     if (ifhwaddr(iface, mac48)) {
00695         return (-1);
00696     }
00697
00698     gettimeofday(&tv, NULL);
00699     ntpts = tvtontp64(&tv);
00700
00701     eui48to64(mac48, eui64);
00702     sha1sum2(sha1, (void *)&ntpts, sizeof(ntpts), (void *)eui64, sizeof(eui64));
00703
00704     prefix[0] = 0xFD; /*0xFC | 0x01 FC00/7 with local bit set [8th bit]*/
00705     memcpy(prefix + 1, sha1+15, 5); /*LSD 40 bits of the SHA hash*/
00706
00707     return (0);
00708 }
00709 #endif
00710
00718 int score_ipv4(struct sockaddr_in *sa4, char *ipaddr, int iplen) {
00719     uint32_t addr;
00720     int nscore;
00721
00722     addr = sa4->sin_addr.s_addr;
00723
00724     /* Get ipaddr string*/
00725     inet_ntop(AF_INET, &sa4->sin_addr, ipaddr, iplen);
00726
00727     /* Score the IP*/
00728     if (!((0xa9fe0000 ^  ntohl(addr)) >> 16)) {
00729         nscore = IPV4_SCORE_ZEROCONF;
00730     } else if (reservedip(ipaddr)) {
00731         nscore = IPV4_SCORE_RESERVED;
00732     } else {
00733         nscore = IPV4_SCORE_ROUTABLE;
00734     }
00735
00736     return nscore;
00737 }
```

```
00738
00746 int score_ipv6(struct sockaddr_in6 *sa6, char *ipaddr, int iplen) {
00747     uint32_t *ipptr, match;
00748     int nscore;
00749
00750 #ifndef __WIN32
00751     ipptr = sa6->sin6_addr.s6_addr32;
00752 #else
00753     ipptr = (uint32_t*)sa6->sin6_addr.u.Word;
00754 #endif
00755     match = ntohl(ipptr[0]) >> 16;
00756
00757     /* exclude link local multicast and special addresses */
00758     if (!(0xFE80 ^ match) || !(0xFF ^ (match >> 8)) || !match) {
00759         return 0;
00760     }
00761
00762     /*Score ip private/sixin4/routable*/
00763     if (!(0xFC ^ (match >> 9))) {
00764         nscore = IPV6_SCORE_RESERVED;
00765     } else if (match == 2002) {
00766         nscore = IPV6_SCORE_SIXIN4;
00767     } else {
00768         nscore = IPV6_SCORE_ROUTABLE;
00769     }
00770     inet_ntop(AF_INET6, ipptr, ipaddr, iplen);
00771
00772     return nscore;
00773 }
00774
00775
00782 #ifndef __WIN32
00783 const char *get_ifipaddr(const char *iface, int family) {
00784     struct ifaddrs *ifaddr, *ifa;
00785     struct sockaddr_in *ipv4addr;
00786     int score = 0, nscore, iflen;
00787     uint32_t subnet = 0, match;
00788     char host[NI_MAXHOST] = "", tmp[NI_MAXHOST];
00789
00790     if (!iface || getifaddrs(&ifaddr) == -1) {
00791         return NULL;
00792     }
00793
00794     for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
00795         iflen = strlen(iface);
00796         if ((ifa->ifa_addr == NULL) ||  strncmp(ifa->ifa_name, iface, iflen) || (ifa->ifa_addr->sa_family
      != family)) {
00797             continue;
00798         }
00799
00800         /* Match aliases not vlans*/
00801         if ((strlen(ifa->ifa_name) > iflen) && (ifa->ifa_name[iflen] != ':')) {
00802             continue;
00803         }
00804
00805         switch (ifa->ifa_addr->sa_family) {
00806             case AF_INET:
00807                 /* Find best ip address for a interface lowest priority is given to zeroconf then reserved
       ip's
00808                  * finally find hte ip with shortest subnet bits.*/
00809                 ipv4addr = (struct sockaddr_in*)ifa->ifa_netmask;
00810                 match = ntohl(~ipv4addr->sin_addr.s_addr);
00811
00812                 nscore = score_ipv4((struct sockaddr_in*)ifa->ifa_addr, tmp, NI_MAXHOST);
00813
00814                 /* match score and subnet*/
00815                 if  ((nscore > score) || ((nscore == score) && (match > subnet))) {
00816                     score = nscore;
00817                     subnet = match;
00818                     strncpy(host, tmp, NI_MAXHOST);
00819                 }
00820                 break;
00821             case AF_INET6:
00822                 nscore = score_ipv6((struct sockaddr_in6*)ifa->ifa_addr, tmp, NI_MAXHOST);
00823
00824                 if (nscore > score) {
00825                     score = nscore;
00826                     strncpy(host, tmp, NI_MAXHOST);
00827                 }
00828                 break;
00829         }
00830     }
00831     freeifaddrs(ifaddr);
00832     return (strlenzero(host)) ? NULL : strdup(host);
00833 }
00834 #endif
```

14.24 src/iputil.c File Reference

IPv4 And IPv6 Utiliies.

```
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <linux/ip.h>
#include <linux/icmp.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <netdb.h>
#include "include/dtsapp.h"
```

Data Structures

- struct pseudohdr

  IPv4 header structur to cast a packet too.

Enumerations

- enum ipversion { IP_PROTO_V4 = 4, IP_PROTO_V6 = 6 }

  IP Protocol numbers.

Functions

- int checkipv6mask (const char *ipaddr, const char *network, uint8_t bits)

  Check if ipaddr is in a network.
- void ipv4tcpchecksum (uint8_t *pkt)

  Update the TCP checksum of a IPv4 packet.
- void ipv4udpchecksum (uint8_t *pkt)

  Update the UDP checksum of a IPv4 packet.
- void ipv4icmpchecksum (uint8_t *pkt)

  Set the checksup of a IPv4 ICMP packet.
- void ipv4checksum (uint8_t *pkt)

  Set the checksup of a IPv4 Packet.
- int packetchecksumv4 (uint8_t *pkt)

  Update the checksum of a IPv4 packet.
- int packetchecksumv6 (uint8_t *pkt)

  Prototype to check checksup on packet.
- int packetchecksum (uint8_t *pkt)

  Generic IPv4 and IPv6 Checksum.
- const char * cidrtosn (int bitlen, char *buf, int size)

  Return the dotted quad notation subnet mask from a CIDR.
- const char * getnetaddr (const char *ipaddr, int cidr, char *buf, int size)

  Return the network address.
- const char * getfirstaddr (const char *ipaddr, int cidr, char *buf, int size)

  Get the first usable address.
- const char * getbcaddr (const char *ipaddr, int cidr, char *buf, int size)

Return broadcast address.

- const char * getlastaddr (const char *ipaddr, int cidr, char *buf, int size)

  Get the last usable address.
- uint32_t cidrcnt (int bitlen)

  Return the number of IP addresses in a given bitmask.
- int reservedip (const char *ipaddr)

  Check IP against list of reserved IP's.
- char * ipv6to4prefix (const char *ipaddr)

  Return IPv6 to IPv4 Prefix fot the address.
- int check_ipv4 (const char *ip, int cidr, const char *test)

  Check if a IP address is in a network.
- void mcast6_ip (struct in6_addr *addr)

  Randomally assign a SSM Multicast address.
  param addr Ip address structure to fill out.
- void mcast4_ip (struct in_addr *addr)

  Randomally assign a SSM Multicast address.
- int inet_lookup (int family, const char *host, void *addr, socklen_t len)

  Perform DNS lookup on a host/ip retun the IP address.

14.24.1 Detailed Description

IPv4 And IPv6 Utiliies.

Definition in file iputil.c.

14.25 iputil.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00023 #include <stdlib.h>
00024 #include <stdint.h>
00025 #include <math.h>
00026 #include <stdio.h>
00027 #include <string.h>
00028 #ifndef __WIN32
00029 #include <linux/ip.h>
00030 #include <linux/icmp.h>
00031 #include <linux/tcp.h>
00032 #include <linux/udp.h>
00033 #include <netdb.h>
00034 #else
00035 #include <winsock2.h>
00036 #include <ws2tcpip.h>
00037 #endif
00038
00039 #include "include/dtsapp.h"
00040
00047 extern int checkipv6mask(const char *ipaddr, const char *network, uint8_t bits) {
00048     uint8_t cnt, bytelen, bitlen;
00049     uint32_t mask, res = 0;
00050     uint32_t *nw = (uint32_t *)network;
```

```
00051     uint32_t *ip = (uint32_t *)ipaddr;
00052
00053     /*calculate significant bytes and bits outside boundry*/
00054     if ((bitlen = bits % 32)) {
00055         bytelen = (bits - bitlen) / 32;
00056         bytelen++;
00057     } else {
00058         bytelen = bits / 32;
00059     }
00060
00061     /*end loop on first mismatch do not check last block*/
00062     for(cnt = 0; (!res && (cnt < (bytelen - 1))); cnt++) {
00063         res += nw[cnt] ^ ip[cnt];
00064     }
00065
00066     /*process last block if no error sofar*/
00067     if (!res) {
00068         mask = (bitlen) ? htonl(~((1 << (32 - bitlen)) - 1)) : -1;
00069         res += (nw[cnt] & mask) ^ (ip[cnt] & mask);
00070     }
00071
00072     return (res);
00073 }
00074
00077 enum ipversion {
00078     IP_PROTO_V4 = 4,
00079     IP_PROTO_V6 = 6
00080 };
00081
00084 struct pseudohdr {
00086     uint32_t saddr;
00088     uint32_t daddr;
00090     uint8_t zero;
00092     uint8_t proto;
00094     uint16_t len;
00095 };
00096
00097 #ifndef __WIN32
00098
00101 extern void ipv4tcpchecksum(uint8_t *pkt) {
00102     struct iphdr *ip = (struct iphdr *)pkt;
00103     struct tcphdr *tcp = (struct tcphdr *)(pkt + (4 * ip->ihl));
00104     uint16_t plen, csum;
00105     struct pseudohdr phdr;
00106
00107     /* get tcp packet len*/
00108     plen = ntohs(ip->tot_len) - (4 * ip->ihl);
00109     tcp->check = 0;
00110     phdr.saddr = ip->saddr;
00111     phdr.daddr = ip->daddr;
00112     phdr.zero = 0;
00113     phdr.proto = ip->protocol;
00114     phdr.len = htons(plen);
00115     csum = checksum(&phdr, sizeof(phdr));
00116     tcp->check = checksum_add(csum, tcp, plen);
00117 }
00118
00122 extern void ipv4udpchecksum(uint8_t *pkt) {
00123     struct iphdr *ip = (struct iphdr *)pkt;
00124     struct udphdr *udp = (struct udphdr *)(pkt + (4 * ip->ihl));
00125     uint16_t csum, plen;
00126     struct pseudohdr phdr;
00127
00128     /* get tcp packet len*/
00129     plen = ntohs(ip->tot_len) - (4 * ip->ihl);
00130     udp->check = 0;
00131     phdr.saddr = ip->saddr;
00132     phdr.daddr = ip->daddr;
00133     phdr.zero = 0;
00134     phdr.proto = ip->protocol;
00135     phdr.len = htons(plen);
00136     csum = checksum(&phdr, sizeof(phdr));
00137     udp->check = checksum_add(csum, udp, plen);
00138 }
00139
00143 extern void ipv4icmpchecksum(uint8_t *pkt) {
00144     struct iphdr *ip = (struct iphdr *)pkt;
00145     struct icmphdr *icmp = (struct icmphdr *)(pkt + (4 * ip->ihl));
00146
00147     icmp->checksum = 0;
00148     icmp->checksum = checksum(icmp, ntohs(ip->tot_len) - (ip->ihl *4));
00149 }
00150
00154 extern void ipv4checksum(uint8_t *pkt) {
00155     struct iphdr *ip = (struct iphdr *)pkt;
00156
00157     ip->check = 0;
```

```
00158     ip->check = checksum(ip, (4 * ip->ihl));
00159 }
00160
00165 extern int packetchecksumv4(uint8_t *pkt) {
00166     struct iphdr *ip = (struct iphdr *)pkt;
00167
00168     ipv4checksum(pkt);
00169
00170     switch(ip->protocol) {
00171         case IPPROTO_ICMP:
00172             ipv4icmpchecksum(pkt);
00173             break;
00174         case IPPROTO_TCP:
00175             ipv4tcpchecksum(pkt);
00176             break;
00177         case IPPROTO_UDP:
00178             ipv4udpchecksum(pkt);
00179             break;
00180         default:
00181             return (-1);
00182     }
00183     return (0);
00184 }
00185
00189 extern int packetchecksumv6(uint8_t *pkt) {
00190     struct iphdr *ip = (struct iphdr *)pkt;
00191     switch(ip->protocol) {
00192         case IPPROTO_ICMP:
00193             break;
00194         case IPPROTO_TCP:
00195             break;
00196         case IPPROTO_UDP:
00197             break;
00198         default:
00199             return (-1);
00200     }
00201     return (0);
00202 }
00203
00208 extern int packetchecksum(uint8_t *pkt) {
00209     struct iphdr *ip = (struct iphdr *)pkt;
00210
00211     switch(ip->version) {
00212         case IP_PROTO_V4:
00213             return (packetchecksumv4(pkt));
00214             break;
00215         case IP_PROTO_V6:
00216             break;
00217     }
00218     return (-1);
00219 }
00220 #endif
00221
00228 extern const char *cidrtosn(int bitlen, char *buf, int size) {
00229     uint32_t nm;
00230     uint8_t *nmb = (uint8_t*)&nm;
00231
00232     if (!buf) {
00233         return NULL;
00234     }
00235
00236     if (bitlen) {
00237         nm = ~((1 << (32-bitlen))-1);
00238     } else {
00239         nm = 0;
00240     }
00241
00242     snprintf(buf, size, "%i.%i.%i.%i", nmb[3], nmb[2], nmb[1], nmb[0]);
00243     return buf;
00244 }
00245
00254 extern const char *getnetaddr(const char *ipaddr, int cidr, char *buf, int size) {
00255     uint32_t ip;
00256     uint8_t *ipb = (uint8_t*)&ip;
00257
00258     if (!buf) {
00259         return NULL;
00260     }
00261
00262 #ifndef __WIN32
00263     inet_pton(AF_INET, ipaddr, &ip);
00264 #else
00265     ip = inet_addr(ipaddr);
00266 #endif
00267     if (cidr) {
00268         ip = ntohl(ip);
00269         ip = ip & ~((1 << (32-cidr))-1);
```

```
00270     } else {
00271         ip = 0;
00272     }
00273 
00274     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00275     return buf;
00276 }
00277 
00286 extern const char *getfirstaddr(const char *ipaddr, int cidr, char *buf, int size) {
00287     uint32_t ip;
00288     uint8_t *ipb = (uint8_t*)&ip;
00289 
00290     if (!buf) {
00291         return NULL;
00292     }
00293 
00294 #ifndef __WIN32
00295     inet_pton(AF_INET, ipaddr, &ip);
00296 #else
00297     ip = inet_addr(ipaddr);
00298 #endif
00299     if (cidr) {
00300         ip = ntohl(ip);
00301         ip = ip & ~((1 << (32-cidr))-1);
00302         ip++;
00303     } else {
00304         ip = 1;
00305     }
00306 
00307     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00308     return buf;
00309 }
00310 
00319 extern const char *getbcaddr(const char *ipaddr, int cidr, char *buf, int size) {
00320     uint32_t ip, mask;
00321     uint8_t *ipb = (uint8_t*)&ip;
00322 
00323 #ifndef __WIN32
00324     inet_pton(AF_INET, ipaddr, &ip);
00325 #else
00326     ip = inet_addr(ipaddr);
00327 #endif
00328     if (cidr) {
00329         mask = (1 << (32-cidr))-1;
00330         ip = ntohl(ip);
00331         ip = (ip & ~mask) | mask;
00332     } else {
00333         ip = 0;
00334     }
00335     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00336     return buf;
00337 }
00338 
00347 extern const char *getlastaddr(const char *ipaddr, int cidr, char *buf, int size) {
00348     uint32_t ip, mask;
00349     uint8_t *ipb = (uint8_t*)&ip;
00350 
00351 #ifndef __WIN32
00352     inet_pton(AF_INET, ipaddr, &ip);
00353 #else
00354     ip = inet_addr(ipaddr);
00355 #endif
00356     if (cidr) {
00357         mask = (1 << (32-cidr))-1;
00358         ip = ntohl(ip);
00359         ip = (ip & ~mask) | mask;
00360         ip--;
00361     } else {
00362         ip = 0;
00363     }
00364     snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
00365     return buf;
00366 }
00367 
00372 extern uint32_t cidrcnt(int bitlen) {
00373     if (bitlen) {
00374         return pow(2, (32-bitlen));
00375     } else {
00376         return 0xFFFFFFFF;
00377     }
00378 }
00379 
00384 extern int reservedip(const char *ipaddr) {
00385     uint32_t ip;
00386 
00387 #ifndef __WIN32
00388     inet_pton(PF_INET, ipaddr, &ip);
```

```
00389 #else
00390     ip = inet_addr(ipaddr);
00391 #endif
00392
00393     ip = ntohl(ip);
00394
00395     if (!((0xe0000000 ^ ip) >> 28)) { /* 224/4*/
00396         return 1;
00397     } else if (!((0x00000000 ^ ip) >> 24)) { /* 0/8 */
00398         return 1;
00399     } else if (!((0x0a000000 ^ ip) >> 24)) { /* 10/8 */
00400         return 1;
00401     } else if (!((0x7f000000 ^ ip) >> 24)) { /* 127/8 */
00402         return 1;
00403     } else if (!((0x64400000 ^ ip) >> 22)) { /* 100.64/10 */
00404         return 1;
00405     } else if (!((0xac100000 ^ ip) >> 20)) { /* 172.16/12 */
00406         return 1;
00407     } else if (!((0xc6120000 ^ ip) >> 17)) { /* 198.18/15 */
00408         return 1;
00409     } else if (!((0xc0a80000 ^ ip) >> 16)) { /* 192.168/16 */
00410         return 1;
00411     } else if (!((0xa9fe0000 ^ ip) >> 16)) { /* 169.254/16 */
00412         return 1;
00413     } else if (!((0xc0000200 ^ ip) >> 8)) { /* 192.0.2/24 */
00414         return 1;
00415     } else if (!((0xc6336400 ^ ip) >> 8)) { /* 198.51.100/24 */
00416         return 1;
00417     } else if (!((0xcb007100 ^ ip) >> 8)) { /* 203.0.113/24 */
00418         return 1;
00419     }
00420     return 0;
00421 }
00422
00427 extern char* ipv6to4prefix(const char *ipaddr) {
00428     uint32_t ip;
00429     uint8_t *ipa;
00430     char *pre6;
00431
00432 #ifndef __WIN32
00433     if (!inet_pton(AF_INET, ipaddr, &ip)) {
00434         return NULL;
00435     }
00436 #else
00437     if (!(ip = inet_addr(ipaddr))) {
00438         return NULL;
00439     }
00440 #endif
00441
00442     pre6 = malloc(10);
00443     ipa=(uint8_t*)&ip;
00444     snprintf(pre6, 10, "%02x%02x:%02x%02x", ipa[0], ipa[1], ipa[2], ipa[3]);
00445     return pre6;
00446 }
00447
00448
00456 extern int check_ipv4(const char* ip, int cidr, const char *test) {
00457     uint32_t ip1, ip2;
00458
00459 #ifndef __WIN32
00460     inet_pton(AF_INET, ip, &ip1);
00461     inet_pton(AF_INET, test, &ip2);
00462 #else
00463     ip1 = inet_addr(ip);
00464     ip2 = inet_addr(test);
00465 #endif
00466
00467     ip1 = ntohl(ip1) >> (32-cidr);
00468     ip2 = ntohl(ip2) >> (32-cidr);
00469
00470     if (!(ip1 ^ ip2)) {
00471         return 1;
00472     } else {
00473         return 0;
00474     }
00475 }
00476
00480 void mcast6_ip(struct in6_addr *addr) {
00481     int mip, rand;
00482     uint32_t *i;
00483
00484 #ifndef __WIN32
00485     i = (uint32_t*)&addr->s6_addr32;
00486 #else
00487     i = (uint32_t*)&addr->u.Word;
00488 #endif
00489     i[0] = htonl(0xFF350000);
```

```
00490     i[1] = 0;
00491     i[2] = 0;
00492     i[3] = 1 << 31;
00493 
00494     do {
00495         rand = genrand(&mip, 4);
00496     } while (!rand);
00497 
00498     i[3] = htonl(i[3] | mip);
00499 }
00500 
00504 void mcast4_ip(struct in_addr *addr) {
00505     uint32_t mip, rand;
00506 
00507     do {
00508         rand = genrand(&mip, 3);
00509         mip >>= 8;
00510     } while (!rand || !(mip >> 8));
00511     mip |= 232 << 24;
00512 
00513     addr->s_addr = htonl(mip);
00514 }
00515 
00523 int inet_lookup(int family, const char *host, void *addr, socklen_t len) {
00524     struct addrinfo hint, *result, *ainfo;
00525     int ret = 0;
00526 
00527     memset(&hint, 0, sizeof(hint));
00528     hint.ai_family = family;
00529 
00530     if (getaddrinfo(host, NULL, &hint, &result) || !result) {
00531         return ret;
00532     }
00533 
00534     for(ainfo = result; ainfo; ainfo = ainfo->ai_next) {
00535         switch(ainfo->ai_family) {
00536             case PF_INET:
00537                 if (len >= sizeof(struct in_addr)) {
00538                     struct sockaddr_in *sa4 = (struct sockaddr_in*)ainfo->ai_addr;
00539                     memcpy(addr, &sa4->sin_addr, len);
00540                     ret = 1;
00541                 }
00542                 break;
00543             case PF_INET6:
00544                 if (len >= sizeof(struct in6_addr)) {
00545                     struct sockaddr_in6 *sa6 = (struct sockaddr_in6*)ainfo->ai_addr;
00546                     memcpy(addr, &sa6->sin6_addr, len);
00547                     ret = 1;
00548                 }
00549                 break;
00550         }
00551         if (ret) {
00552             break;
00553         }
00554     }
00555     freeaddrinfo(result);
00556     return ret;
00557 }
```

14.26 src/libxml2.c File Reference

XML Interface.

```
#include <string.h>
#include <stdint.h>
#include <libxml/tree.h>
#include <libxml/parser.h>
#include <libxml/xpath.h>
#include <libxml/xpathInternals.h>
#include "include/priv_xml.h"
#include "include/dtsapp.h"
```

Data Structures

- struct xml_node_iter

  Iterator to traverse nodes in a xpath.
- struct xml_search

  XML xpath search result.

Functions

- void xml_free_buffer (void *data)

  Reference destructor for xml_buffer.
- struct xml_doc * xml_loaddoc (const char *docfile, int validate)

  Load a XML file into XML document and return reference.
- struct xml_doc * xml_loadbuf (const uint8_t *buffer, uint32_t len, int validate)

  Load a buffer into XML document returning refereence.
- struct xml_node * xml_getrootnode (struct xml_doc *xmldoc)

  Return reference to the root node.
- struct xml_node * xml_getfirstnode (struct xml_search *xpsearch, void **iter)

  Return reference to the first node optionally creating a iterator.
- struct xml_node * xml_getnextnode (void *iter)

  Return the next node.
- struct bucket_list * xml_getnodes (struct xml_search *xpsearch)

  Return reference to bucket list containing nodes.
- struct xml_search * xml_xpath (struct xml_doc *xmldata, const char *xpath, const char *attrkey)

  Return a reference to a xpath search result.
- int xml_nodecount (struct xml_search *xsearch)

  Return the number of nodes in the search path.
- struct xml_node * xml_getnode (struct xml_search *xsearch, const char *key)

  Return a node in the search matching key.
- const char * xml_getattr (struct xml_node *xnode, const char *attr)

  Return value of attribute.
- const char * xml_getrootname (struct xml_doc *xmldoc)

  Return the name of the root node.
- void xml_modify (struct xml_doc *xmldoc, struct xml_node *xnode, const char *value)

  Modify a XML node.
- void xml_setattr (struct xml_doc *xmldoc, struct xml_node *xnode, const char *name, const char *value)

  Modify a XML node attribute.
- void xml_createpath (struct xml_doc *xmldoc, const char *xpath)

  Create a path in XML document.
- void xml_appendnode (struct xml_doc *xmldoc, const char *xpath, struct xml_node *child)

  Append a node to a path.
- struct xml_node * xml_addnode (struct xml_doc *xmldoc, const char *xpath, const char *name, const char *value, const char *attrkey, const char *keyval)

  Append a node to a path.
- void xml_unlink (struct xml_node *xnode)

  Unlink a node from the document.
- void xml_delete (struct xml_node *xnode)

  Delete a node from document it is not unrefd and should be.
- char * xml_getbuffer (void *buffer)

  Return the buffer of a xml_buffer structure.
- void * xml_doctobuffer (struct xml_doc *xmldoc)

Return a dump of a XML document.

- void xml_init ()

  Initialise/Reference the XML library.

- void xml_close ()

  Unreference the XML library.

- void xml_savefile (struct xml_doc *xmldoc, const char *file, int format, int compress)

  Save XML document to a file.

14.26.1 Detailed Description

XML Interface.

Definition in file libxml2.c.

14.27 libxml2.c

```
00001
00007 #include <string.h>
00008 #include <stdint.h>
00009 #ifdef __WIN32__
00010 #include <sec_api/string_s.h>
00011 #endif
00012
00013 #include <libxml/tree.h>
00014 #include <libxml/parser.h>
00015 #include <libxml/xpath.h>
00016 #include <libxml/xpathInternals.h>
00017
00018 #include "include/priv_xml.h"
00019 #include "include/dtsapp.h"
00020
00022 struct xml_node_iter {
00024     struct xml_search *xsearch;
00026     int curpos;
00028     int cnt;
00029 };
00030
00033 struct xml_search {
00035     struct xml_doc *xmldoc;
00037     xmlXPathObjectPtr xpathObj;
00039     struct bucket_list *nodes;
00040 };
00041
00042 static void *xml_has_init_parser = NULL;
00043
00046 void xml_free_buffer(void *data) {
00047     struct xml_buffer *xb = data;
00048     xmlFree(xb->buffer);
00049 }
00050
00051 static void free_xmlsearch(void *data) {
00052     struct xml_search *xs = data;
00053     objunref(xs->xmldoc);
00054     objunref(xs->nodes);
00055     xmlXPathFreeObject(xs->xpathObj);
00056 }
00057
00058 static void free_parser(void *data) {
00059     xmlCleanupParser();
00060 }
00061
00062 static void free_xmlnode(void *data) {
00063     struct xml_node *ninfo = data;
00064
00065     if (ninfo->attrs) {
00066         objunref(ninfo->attrs);
00067     }
00068     if (ninfo->name) {
00069         free((char *)ninfo->name);
00070     }
00071     if (ninfo->key) {
00072         free((char *)ninfo->key);
00073     }
00074     if (ninfo->value) {
00075         free((char *)ninfo->value);
```

```
00076     }
00077 }
00078
00079 static void free_xmldata(void *data) {
00080     struct xml_doc *xmldata = data;
00081
00082     if (xmldata->xpathCtx) {
00083         xmlXPathFreeContext(xmldata->xpathCtx);
00084     }
00085     if (xmldata->doc) {
00086         xmlFreeDoc(xmldata->doc);
00087     }
00088     if (xmldata->ValidCtxt) {
00089         xmlFreeValidCtxt(xmldata->ValidCtxt);
00090     }
00091     xml_close();
00092 }
00093
00094 static int32_t node_hash(const void *data, int key) {
00095     int ret;
00096     const struct xml_node *ni = data;
00097     const char *hashkey = (key) ? data : ni->key;
00098
00099     if (hashkey) {
00100         ret = jenhash(hashkey, strlen(hashkey), 0);
00101     } else {
00102         ret = jenhash(ni, sizeof(ni), 0);
00103     }
00104     return(ret);
00105 }
00106
00107 static int32_t attr_hash(const void *data, int key) {
00108     int ret;
00109     const struct xml_attr *ai = data;
00110     const char *hashkey = (key) ? data : ai->name;
00111
00112     ret = jenhash(hashkey, strlen(hashkey), 0);
00113
00114     return(ret);
00115 }
00116
00117 static struct xml_doc *xml_setup_parse(struct xml_doc *xmldata, int validate) {
00118     if (validate) {
00119         if (!(xmldata->ValidCtxt = xmlNewValidCtxt())) {
00120             objunref(xmldata);
00121             return NULL;
00122         }
00123         if (!xmlValidateDocument(xmldata->ValidCtxt, xmldata->doc)) {
00124             objunref(xmldata);
00125             return NULL;
00126         }
00127         /*xmlValidateDocumentFinal(xmldata->ValidCtxt, xmldata->doc);*/
00128     }
00129
00130     if (!(xmldata->root = xmlDocGetRootElement(xmldata->doc))) {
00131         objunref(xmldata);
00132         return NULL;
00133     }
00134
00135     if (!(xmldata->xpathCtx = xmlXPathNewContext(xmldata->doc))) {
00136         objunref(xmldata);
00137         return NULL;
00138     }
00139     return xmldata;
00140 }
00141
00146 extern struct xml_doc *xml_loaddoc(const char *docfile, int validate) {
00147     struct xml_doc *xmldata;
00148
00149     xml_init();
00150
00151     if (!(xmldata = objalloc(sizeof(*xmldata), free_xmldata))) {
00152         return NULL;
00153     }
00154
00155     if (!(xmldata->doc = xmlParseFile(docfile))) {
00156         objunref(xmldata);
00157         return NULL;
00158     }
00159
00160     return xml_setup_parse(xmldata, validate);
00161 }
00162
00168 extern struct xml_doc *xml_loadbuf(const uint8_t *buffer, uint32_t len, int validate) {
00169     struct xml_doc *xmldata;
00170     int flags;
00171
```

```
00172     xml_init();
00173
00174     if (!(xmldata = objalloc(sizeof(*xmldata), free_xmldata))) {
00175         return NULL;
00176     }
00177
00178     if (validate) {
00179         flags = XML_PARSE_DTDLOAD | XML_PARSE_DTDVALID;
00180     } else {
00181         flags = XML_PARSE_DTDVALID;
00182     }
00183
00184     if (!(xmldata->doc = xmlReadMemory((const char *)buffer, len, NULL, NULL, flags))) {
00185         objunref(xmldata);
00186         return NULL;
00187     }
00188     return xml_setup_parse(xmldata, 0);
00189 }
00190
00191 static struct xml_node *xml_nodetohash(struct xml_doc *xmldoc, xmlNodePtr node, const char *
      attrkey) {
00192     struct xml_node *ninfo;
00193     struct xml_attr *ainfo;
00194     xmlChar *xmlstr;
00195     xmlAttr *attrs;
00196
00197     if (!(ninfo = objalloc(sizeof(*ninfo), free_xmlnode))) {
00198         return NULL;
00199     }
00200     ninfo->attrs = NULL;
00201
00202     if (!(ninfo->attrs = create_bucketlist(0, attr_hash))) {
00203         objunref(ninfo);
00204         return NULL;
00205     }
00206
00207     ALLOC_CONST(ninfo->name, (const char *)node->name);
00208     xmlstr = xmlNodeListGetString(xmldoc->doc, node->xmlChildrenNode, 1);
00209     ALLOC_CONST(ninfo->value, (const char *)xmlstr);
00210     xmlFree(xmlstr);
00211     ninfo->nodeptr = node;
00212
00213     attrs = node->properties;
00214     while(attrs && attrs->name && attrs->children) {
00215         if (!(ainfo = objalloc(sizeof(*ainfo), NULL))) {
00216             objunref(ninfo);
00217             return NULL;
00218         }
00219         ALLOC_CONST(ainfo->name, (const char *)attrs->name);
00220         xmlstr = xmlNodeListGetString(xmldoc->doc, attrs->children, 1);
00221         ALLOC_CONST(ainfo->value, (const char *)xmlstr);
00222         if (attrkey && !strcmp((const char *)attrs->name, (const char *)attrkey)) {
00223             ALLOC_CONST(ninfo->key, (const char *)xmlstr);
00224         }
00225         xmlFree(xmlstr);
00226         addtobucket(ninfo->attrs, ainfo);
00227         objunref(ainfo);
00228         attrs = attrs->next;
00229     }
00230     if (!attrkey && ninfo->value) {
00231         ALLOC_CONST(ninfo->key, ninfo->value);
00232     }
00233     return ninfo;
00234 }
00235
00236 static struct xml_node *xml_gethash(struct xml_search *xpsearch, int i, const char *
      attrkey) {
00237     xmlNodePtr node;
00238     xmlNodeSetPtr nodeset;
00239     struct xml_node *xn;
00240
00241     if (!objref(xpsearch)) {
00242         return NULL;
00243     }
00244
00245     objlock(xpsearch->xmldoc);
00246     objlock(xpsearch);
00247     if (!(nodeset = xpsearch->xpathObj->nodesetval)) {
00248         objunlock(xpsearch);
00249         objunlock(xpsearch->xmldoc);
00250         objunref(xpsearch);
00251         return NULL;
00252     }
00253
00254     if (!(node = nodeset->nodeTab[i])) {
00255         objunlock(xpsearch);
00256         objunlock(xpsearch->xmldoc);
```

```
00257         objunref(xpsearch);
00258         return NULL;
00259     }
00260     xn = xml_nodetohash(xpsearch->xmldoc, node, attrkey);
00261     objunlock(xpsearch);
00262     objunlock(xpsearch->xmldoc);
00263     objunref(xpsearch);
00264 
00265     return xn;
00266 }
00267 
00268 static void free_iter(void *data) {
00269     struct xml_node_iter *xi = data;
00270 
00271     objunref(xi->xsearch);
00272 }
00273 
00276 extern struct xml_node *xml_getrootnode(struct xml_doc *xmldoc) {
00277     struct xml_node *rn;
00278 
00279     objlock(xmldoc);
00280     rn = xml_nodetohash(xmldoc, xmldoc->root, NULL);
00281     objunlock(xmldoc);
00282     return rn;
00283 }
00284 
00295 extern struct xml_node *xml_getfirstnode(struct
    xml_search *xpsearch, void **iter) {
00296     struct xml_node_iter *newiter;
00297     struct xml_node *xn;
00298 
00299     if (!objref(xpsearch)) {
00300         return NULL;
00301     }
00302 
00303     if (iter) {
00304         newiter = objalloc(sizeof(*newiter), free_iter);
00305         objlock(xpsearch);
00306         newiter->cnt = xml_nodecount(xpsearch);
00307         objunlock(xpsearch);
00308         newiter->curpos = 0;
00309         newiter->xsearch = xpsearch;
00310         objref(newiter->xsearch);
00311         *iter = newiter;
00312     }
00313 
00314     xn = xml_gethash(xpsearch, 0, NULL);
00315     objunref(xpsearch);
00316     return xn;
00317 }
00318 
00322 extern struct xml_node *xml_getnextnode(void *iter) {
00323     struct xml_node_iter *xi = iter;
00324     struct xml_node *xn;
00325 
00326     if (!objref(xi->xsearch)) {
00327         return NULL;
00328     }
00329 
00330     objlock(xi);
00331     xi->curpos ++;
00332     if (xi->curpos >= xi->cnt) {
00333         objunlock(xi);
00334         objunref(xi->xsearch);
00335         return NULL;
00336     }
00337     xn = xml_gethash(xi->xsearch, xi->curpos, NULL);
00338     objunlock(xi);
00339     objunref(xi->xsearch);
00340 
00341     return xn;
00342 }
00343 
00349 extern struct bucket_list *xml_getnodes(struct xml_search *xpsearch) {
00350     return (xpsearch && objref(xpsearch->nodes)) ? xpsearch->nodes : NULL;
00351 }
00352 
00353 static struct bucket_list *xml_setnodes(struct xml_search *xpsearch, const char *
    attrkey) {
00354     struct xml_node *ninfo;
00355     struct bucket_list *nodes;
00356     int cnt, i;
00357 
00358     if (!(nodes = create_bucketlist(2, node_hash))) {
00359         return NULL;
00360     }
00361 
```

```
00362     cnt = xml_nodecount(xpsearch);
00363     for(i=0; i < cnt; i++) {
00364         ninfo = xml_gethash(xpsearch, i, attrkey);
00365         if (!addtobucket(nodes, ninfo)) {
00366             objunref(ninfo);
00367             objunref(nodes);
00368             nodes = NULL;
00369             break;
00370         }
00371         objunref(ninfo);
00372     }
00373     return nodes;
00374 }
00375
00381 extern struct xml_search *xml_xpath(struct xml_doc *xmldata, const char *xpath,
      const char *attrkey) {
00382     struct xml_search *xpsearch;
00383
00384     if (!objref(xmldata) || !(xpsearch = objalloc(sizeof(*xpsearch), free_xmlsearch))) {
00385         return NULL;
00386     }
00387
00388     objlock(xmldata);
00389     xpsearch->xmldoc = xmldata;
00390     if (!(xpsearch->xpathObj = xmlXPathEvalExpression((const xmlChar *)xpath, xmldata->xpathCtx)))
      {
00391         objunlock(xmldata);
00392         objunref(xpsearch);
00393         return NULL;
00394     }
00395
00396     if (xmlXPathNodeSetIsEmpty(xpsearch->xpathObj->nodesetval)) {
00397         objunlock(xmldata);
00398         objunref(xpsearch);
00399         return NULL;
00400     }
00401     objunlock(xmldata);
00402
00403     if (!(xpsearch->nodes = xml_setnodes(xpsearch, attrkey))) {
00404         objunref(xpsearch);
00405         return NULL;
00406     }
00407     return xpsearch;
00408 }
00409
00413 extern int xml_nodecount(struct xml_search *xsearch) {
00414     xmlNodeSetPtr nodeset;
00415
00416     if (xsearch && xsearch->xpathObj && ((nodeset = xsearch->xpathObj->nodesetval))) {
00417         return nodeset->nodeNr;
00418     } else {
00419         return 0;
00420     }
00421 }
00422
00429 extern struct xml_node *xml_getnode(struct xml_search *xsearch, const char *
      key) {
00430     if (!xsearch) {
00431         return NULL;
00432     }
00433     return bucket_list_find_key(xsearch->nodes, key);
00434 }
00435
00440 extern const char *xml_getattr(struct xml_node *xnode, const char *attr) {
00441     struct xml_attr *ainfo;
00442
00443     if (!xnode) {
00444         return NULL;
00445     }
00446
00447     if ((ainfo = bucket_list_find_key(xnode->attrs, attr))) {
00448         objunref(ainfo);
00449         return ainfo->value;
00450     } else {
00451         return NULL;
00452     }
00453 }
00454
00458 extern const char *xml_getrootname(struct xml_doc *xmldoc) {
00459     if (xmldoc) {
00460         return (const char *)xmldoc->root->name;
00461     }
00462     return NULL;
00463 }
00464
00469 extern void xml_modify(struct xml_doc *xmldoc, struct xml_node *xnode, const char
      *value) {
```

```
00470     xmlChar *encval;
00471     xmlNodePtr node;
00472
00473     objlock(xmldoc);
00474     node = xnode->nodeptr;
00475     encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
00476     xmlNodeSetContent(node, encval);
00477     xmlFree(encval);
00478     encval = xmlNodeListGetString(xmldoc->doc, node->xmlChildrenNode, 1);
00479     objunlock(xmldoc);
00480
00481     if (xnode->value) {
00482         free((void*)xnode->value);
00483     }
00484     ALLOC_CONST(xnode->value, (const char *)encval);
00485     xmlFree(encval);
00486 }
00487
00493 extern void xml_setattr(struct xml_doc *xmldoc, struct
      xml_node *xnode, const char *name, const char *value) {
00494     xmlChar *encval;
00495
00496     objlock(xmldoc);
00497     encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
00498     xmlSetProp(xnode->nodeptr, (const xmlChar *)name, (const xmlChar *)encval);
00499     objunlock(xmldoc);
00500     xmlFree(encval);
00501 }
00502
00507 extern void xml_createpath(struct xml_doc *xmldoc, const char *xpath) {
00508     struct xml_node *nn;
00509     xmlXPathObjectPtr xpathObj;
00510     char *lpath, *tok, *save, *cpath, *dup;
00511     const char *root = (char *)xmldoc->root->name;
00512     int len;
00513
00514
00515     if (!objref(xmldoc)) {
00516         return;
00517     }
00518
00519     if (!(dup = strdup(xpath))) {
00520         objunref(xmldoc);
00521         return;
00522     }
00523
00524     len = strlen(xpath)+1;
00525     if (!(cpath = malloc(len))) {
00526         free(dup);
00527         objunref(xmldoc);
00528         return;
00529     }
00530     if (!(lpath = malloc(len))) {
00531         free(dup);
00532         free(cpath);
00533         objunref(xmldoc);
00534         return;
00535     }
00536
00537     cpath[0] = '\0';
00538     lpath[0] = '\0';
00539
00540 #ifndef __WIN32__
00541         for (tok = strtok_r(dup, "/", &save); tok ; tok = strtok_r(NULL, "/", &save)) {
00542 #else
00543         for (tok = strtok_s(dup, "/", &save); tok ; tok = strtok_s(NULL, "/", &save)) {
00544 #endif
00545         strcat(cpath,"/");
00546         strcat(cpath, tok);
00547         if (!strcmp(tok, root)) {
00548             strcat(lpath,"/");
00549             strcat(lpath, tok);
00550             continue;
00551         }
00552
00553         objlock(xmldoc);
00554         if (!(xpathObj = xmlXPathEvalExpression((const xmlChar *)cpath, xmldoc->xpathCtx))) {
00555             objunlock(xmldoc);
00556             free(lpath);
00557             free(cpath);
00558             free(dup);
00559             objunref(xmldoc);
00560             return;
00561         }
00562         objunlock(xmldoc);
00563
00564         if (xmlXPathNodeSetIsEmpty(xpathObj->nodesetval)) {
```

```
00565             nn = xml_addnode(xmldoc, lpath, tok, NULL, NULL, NULL);
00566             objunref(nn);
00567         }
00568
00569         xmlXPathFreeObject(xpathObj);
00570         strcat(lpath,"/");
00571         strcat(lpath, tok);
00572     }
00573
00574     free(dup);
00575     free(lpath);
00576     free(cpath);
00577     objunref(xmldoc);
00578 }
00579
00580
00581 static xmlNodePtr xml_getparent(struct xml_doc *xmldoc, const char *xpath) {
00582     xmlXPathObjectPtr xpathObj;
00583     xmlNodePtr parent = NULL;
00584     xmlNodeSetPtr nodes;
00585     int i, cnt;
00586
00587     if (!(xpathObj = xmlXPathEvalExpression((const xmlChar *)xpath, xmldoc->xpathCtx))) {
00588         return NULL;
00589     }
00590
00591     if (xmlXPathNodeSetIsEmpty(xpathObj->nodesetval)) {
00592         xmlXPathFreeObject(xpathObj);
00593         return NULL;
00594     }
00595
00596     if (!(nodes = xpathObj->nodesetval)) {
00597         xmlXPathFreeObject(xpathObj);
00598         return NULL;
00599     }
00600
00601     cnt = nodes->nodeNr;
00602     for(i=cnt - 1; i >= 0; i--) {
00603         if (nodes->nodeTab[i]->type == XML_ELEMENT_NODE) {
00604             parent=nodes->nodeTab[i];
00605             nodes->nodeTab[i] = NULL;
00606             break;
00607         }
00608     }
00609
00610     if (!parent) {
00611         xmlXPathFreeObject(xpathObj);
00612         return NULL;
00613     }
00614
00615     xmlXPathFreeObject(xpathObj);
00616     return parent;
00617 }
00618
00619
00625 extern void xml_appendnode(struct xml_doc *xmldoc, const char *xpath, struct
      xml_node *child) {
00626     xmlNodePtr parent;
00627
00628     if (!objref(xmldoc)) {
00629         return;
00630     }
00631
00632     objlock(xmldoc);
00633     if (!(parent = xml_getparent(xmldoc, xpath))) {
00634         objunlock(xmldoc);
00635         objunref(xmldoc);
00636     }
00637
00638     xmlAddChild(parent,child->nodeptr);
00639     objunlock(xmldoc);
00640     objunref(xmldoc);
00641 }
00642
00651 extern struct xml_node *xml_addnode(struct xml_doc *xmldoc, const char *xpath,
      const char *name, const char *value,
00652                                     const char *attrkey, const char *keyval) {
00653     struct xml_node *newnode;
00654     xmlNodePtr parent;
00655     xmlNodePtr child;
00656     xmlChar *encval;
00657
00658     if (!objref(xmldoc)) {
00659         return NULL;
00660     }
00661
00662     objlock(xmldoc);
```

```
00663     if (!(parent = xml_getparent(xmldoc, xpath))) {
00664         objunlock(xmldoc);
00665         objunref(xmldoc);
00666         return NULL;
00667     }
00668 
00669     encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
00670     child = xmlNewDocNode(xmldoc->doc, NULL, (const xmlChar *)name, encval);
00671     xmlFree(encval);
00672     xmlAddChild(parent,child);
00673 
00674     if (attrkey && keyval) {
00675         encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)keyval);
00676         xmlSetProp(child, (const xmlChar *)attrkey, (const xmlChar *)encval);
00677         xmlFree(encval);
00678     }
00679     objunlock(xmldoc);
00680 
00681     if (!(newnode = xml_nodetohash(xmldoc, child, attrkey))) {
00682         objunref(xmldoc);
00683         return NULL;
00684     }
00685 
00686     objunref(xmldoc);
00687 
00688     return newnode;
00689 }
00690 
00693 extern void xml_unlink(struct xml_node *xnode) {
00694     objlock(xnode);
00695     xmlUnlinkNode(xnode->nodeptr);
00696     objunlock(xnode);
00697 }
00698 
00701 extern void xml_delete(struct xml_node *xnode) {
00702     objlock(xnode);
00703     xmlUnlinkNode(xnode->nodeptr);
00704     xmlFreeNode(xnode->nodeptr);
00705     xnode->nodeptr = NULL;
00706     objunlock(xnode);
00707 }
00708 
00712 extern char *xml_getbuffer(void *buffer) {
00713     struct xml_buffer *xb = buffer;
00714 
00715     if (!xb) {
00716         return NULL;
00717     }
00718     return (char *)xb->buffer;
00719 }
00720 
00726 extern void *xml_doctobuffer(struct xml_doc *xmldoc) {
00727     struct xml_buffer *xmlbuf;
00728 
00729     if (!(xmlbuf = objalloc(sizeof(*xmlbuf),xml_free_buffer))) {
00730         return NULL;
00731     }
00732 
00733     objlock(xmldoc);
00734     xmlDocDumpFormatMemory(xmldoc->doc, &xmlbuf->buffer, &xmlbuf->size, 1);
00735     objunlock(xmldoc);
00736     return xmlbuf;
00737 }
00738 
00742 extern void xml_init() {
00743     if (!xml_has_init_parser) {
00744         xml_has_init_parser = objalloc(0, free_parser);
00745         xmlInitParser();
00746         LIBXML_TEST_VERSION
00747         xmlKeepBlanksDefault(0);
00748         xmlLoadExtDtdDefaultValue = 1;
00749         xmlSubstituteEntitiesDefault(1);
00750     } else {
00751         objref(xml_has_init_parser);
00752     }
00753 }
00754 
00758 extern void xml_close() {
00759     if (xml_has_init_parser) {
00760         objunref(xml_has_init_parser);
00761     }
00762 }
00763 
00769 extern void xml_savefile(struct xml_doc *xmldoc, const char *file, int format, int
      compress) {
00770     objlock(xmldoc);
00771     xmlSetDocCompressMode(xmldoc->doc, compress);
```

```
00772     xmlSaveFormatFile(file, xmldoc->doc, format);
00773     xmlSetDocCompressMode(xmldoc->doc, 0);
00774     objunlock(xmldoc);
00775 }
00776
00777 /*static void xml_modify2(struct xml_search *xpsearch, struct xml_node *xnode, const char *value) {
00778     xmlNodeSetPtr nodes;
00779     int size, i;
00780
00781     if (!(nodes = xpsearch->xpathObj->nodesetval)) {
00782         return;
00783     }
00784
00785     size = (nodes) ? nodes->nodeNr : 0;
00786
00787 */  /*
00788      * http://www.xmlsoft.org/examples/xpath2.c
00789      * remove the reference to the modified nodes from the node set
00790      * as they are processed, if they are not namespace nodes.
00791     */
00792 /*  for(i = size - 1; i >= 0; i--) {
00793         if (nodes->nodeTab[i] == xnode->nodeptr) {
00794             xmlNodeSetContent(nodes->nodeTab[i], (const xmlChar *)value);
00795             if (nodes->nodeTab[i]->type != XML_NAMESPACE_DECL) {
00796                 nodes->nodeTab[i] = NULL;
00797             }
00798         }
00799     }
00800 }*/
00801
```

14.28 src/libxslt.c File Reference

XSLT Interface.

```
#include <stdint.h>
#include <string.h>
#include <libxslt/xsltutils.h>
#include <libxslt/transform.h>
#include "include/dtsapp.h"
#include "include/priv_xml.h"
```

Data Structures

- struct xslt_doc

  XSLT Document.
- struct xslt_param

  XSLT Parameter name/value pair.

Functions

- struct xslt_doc * xslt_open (const char *xsltfile)

  Open a XSLT file returning reference to it.
- void xslt_addparam (struct xslt_doc *xsltdoc, const char *param, const char *value)

  Add a parameter to the XSLT document.
- void xslt_clearparam (struct xslt_doc *xsltdoc)

  Delete all parameters of a XSLT document.
- void xslt_apply (struct xml_doc *xmldoc, struct xslt_doc *xsltdoc, const char *filename, int comp)

  Apply XSLT document to a XML document.
- void * xslt_apply_buffer (struct xml_doc *xmldoc, struct xslt_doc *xsltdoc)

  Apply XSLT document to a XML document returning result in buffer.
- void xslt_init ()

Reference the XSLT parser.

- void xslt_close ()

  Release reference to XSLT parser.

14.28.1 Detailed Description

XSLT Interface.

Definition in file libxslt.c.

14.29 libxslt.c

```
00001
00007 #include <stdint.h>
00008 #ifdef __WIN32__
00009 #include <winsock2.h>
00010 #include <windows.h>
00011 #endif
00012 #include <string.h>
00013
00014 #include <libxslt/xsltutils.h>
00015 #include <libxslt/transform.h>
00016
00017 #include "include/dtsapp.h"
00018 #include "include/priv_xml.h"
00019
00021 struct xslt_doc {
00023     xsltStylesheetPtr doc;
00025     struct bucket_list *params;
00026 };
00027
00029 struct xslt_param {
00031     const char *name;
00033     const char *value;
00034 };
00035
00036 static void *xslt_has_init_parser = NULL;
00037
00038 static void free_xsltdoc(void *data) {
00039     struct xslt_doc *xsltdoc = data;
00040
00041     xsltFreeStylesheet(xsltdoc->doc);
00042     objunref(xsltdoc->params);
00043     xslt_close();
00044 }
00045
00046 static void free_parser(void *data) {
00047     xsltCleanupGlobals();
00048     xmlCleanupParser();
00049 }
00050
00051 static int32_t xslt_hash(const void *data, int key) {
00052     int ret;
00053     const struct xslt_param *xp = data;
00054     const char *hashkey = (key) ? data : xp->name;
00055
00056     if (hashkey) {
00057         ret = jenhash(hashkey, strlen(hashkey), 0);
00058     } else {
00059         ret = jenhash(xp, sizeof(xp), 0);
00060     }
00061     return(ret);
00062 }
00063
00067 extern struct xslt_doc *xslt_open(const char *xsltfile) {
00068     struct xslt_doc *xsltdoc;
00069
00070     if (!(xsltdoc = objalloc(sizeof(*xsltdoc), free_xsltdoc))) {
00071         return NULL;
00072     }
00073     xslt_init();
00074
00075     xsltdoc->doc = xsltParseStylesheetFile((const xmlChar *)xsltfile);
00076     xsltdoc->params = create_bucketlist(0, xslt_hash);
00077     return xsltdoc;
00078 }
00079
00080 static void free_param(void *data) {
```

```
00081     struct xslt_param *param = data;
00082     if (param->name) {
00083         free((void *)param->name);
00084     }
00085     if (param->value) {
00086         free((void *)param->value);
00087     }
00088 }
00089
00094 extern void xslt_addparam(struct xslt_doc *xsltdoc, const char *param, const char *
      value) {
00095     struct xslt_param *xparam;
00096     int size;
00097
00098     if (!xsltdoc || !xsltdoc->params || !objref(xsltdoc) || !(xparam =
      objalloc(sizeof(*xparam), free_param))) {
00099         return;
00100     }
00101
00102     size = strlen(value) + 3;
00103     ALLOC_CONST(xparam->name, param);
00104     xparam->value = malloc(size);
00105     snprintf((char *)xparam->value, size, "'%s'", value);
00106     objlock(xsltdoc);
00107     addtobucket(xsltdoc->params, xparam);
00108     objunlock(xsltdoc);
00109     objunref(xparam);
00110     objunref(xsltdoc);
00111 }
00112
00115 void xslt_clearparam(struct xslt_doc *xsltdoc) {
00116     if (!xsltdoc || !xsltdoc->params) {
00117         return;
00118     }
00119
00120     objlock(xsltdoc);
00121     objunref(xsltdoc->params);
00122     xsltdoc->params = create_bucketlist(0, xslt_hash);
00123     objunlock(xsltdoc);
00124 }
00125
00126 /* grabs ref to xmldoc/xsltdoc and locks xsltdoc*/
00127 static const char **xslt_params(struct xml_doc *xmldoc, struct xslt_doc *xsltdoc) {
00128     const char **params = NULL;
00129     struct xslt_param *xparam;
00130     struct bucket_loop *bloop;
00131     int cnt=0;
00132
00133     if (!objref(xmldoc)) {
00134         return NULL;
00135     }
00136
00137     if (!objref(xsltdoc)) {
00138         objunref(xmldoc);
00139         return NULL;
00140     }
00141
00142     objlock(xsltdoc);
00143     if (!(params = malloc(sizeof(void *) * (bucket_list_cnt(xsltdoc->
      params)*2 + 2)))) {
00144         objunlock(xsltdoc);
00145         objunref(xsltdoc);
00146         objunref(xmldoc);
00147         return NULL;
00148     }
00149
00150     bloop = init_bucket_loop(xsltdoc->params);
00151     while(bloop && (xparam = next_bucket_loop(bloop))) {
00152         params[cnt] = xparam->name;
00153         cnt++;
00154         params[cnt] = xparam->value;
00155         cnt++;
00156         objunref(xparam);
00157     };
00158     params[cnt] = NULL;
00159     return params;
00160 }
00161
00167 extern void xslt_apply(struct xml_doc *xmldoc, struct xslt_doc *xsltdoc, const
      char *filename, int comp) {
00168     const char **params = NULL;
00169     xmlDocPtr res;
00170
00171     /* ref's xml/xslt locks xslt IF set*/
00172     if (!(params = xslt_params(xmldoc, xsltdoc))) {
00173         return;
00174     }
```

```
00175
00176 #ifndef __WIN32__
00177     touch(filename, 80, 80);
00178 #else
00179     touch(filename);
00180 #endif
00181     objlock(xmldoc);
00182     res = xsltApplyStylesheet(xsltdoc->doc, xmldoc->doc, params);
00183     xsltSaveResultToFilename(filename, res, xsltdoc->doc, comp);
00184     objunlock(xmldoc);
00185     objunref(xmldoc);
00186     objunlock(xsltdoc);
00187
00188     free(params);
00189     xmlFreeDoc(res);
00190     xslt_clearparam(xsltdoc);
00191     objunref(xsltdoc);
00192 }
00193
00198 extern void *xslt_apply_buffer(struct xml_doc *xmldoc, struct
      xslt_doc *xsltdoc) {
00199     struct xml_buffer *xmlbuf;
00200     const char **params;
00201     xmlDocPtr res;
00202
00203     if (!(xmlbuf = objalloc(sizeof(*xmlbuf),xml_free_buffer))) {
00204         return NULL;
00205     }
00206
00207     if (!(params = xslt_params(xmldoc, xsltdoc))) {
00208         objunref(xmlbuf);
00209         return NULL;
00210     }
00211
00212     objlock(xmldoc);
00213     res = xsltApplyStylesheet(xsltdoc->doc, xmldoc->doc, params);
00214     xsltSaveResultToString(&xmlbuf->buffer, &xmlbuf->size, res, xsltdoc->doc);
00215     objunlock(xmldoc);
00216     objunref(xmldoc);
00217     objunlock(xsltdoc);
00218
00219     free(params);
00220     xmlFreeDoc(res);
00221     xslt_clearparam(xsltdoc);
00222     objunref(xsltdoc);
00223
00224     return xmlbuf;
00225 }
00226
00230 extern void xslt_init() {
00231     if (!xslt_has_init_parser) {
00232         xslt_has_init_parser=objalloc(0, free_parser);
00233     } else {
00234         objref(xslt_has_init_parser);
00235     }
00236 }
00237
00241 extern void xslt_close() {
00242     if (xslt_has_init_parser) {
00243         objunref(xslt_has_init_parser);
00244     }
00245 }
00246
```

14.30 src/lookup3.c File Reference

by Bob Jenkins, May 2006, Public Domain.

```
#include <stdio.h>
#include <time.h>
#include <stdint.h>
#include <sys/param.h>
```

Macros

- #define HASH_LITTLE_ENDIAN 0

- #define HASH_BIG_ENDIAN 0
- #define hashsize(n) ((uint32_t)1<<(n))
- #define hashmask(n) (hashsize(n)-1)
- #define rot(x, k) (((x)<<(k)) | ((x)>>(32-(k))))
- #define mix(a, b, c)

  mix 3 32-bit values reversibly
- #define final(a, b, c)

  final mixing of 3 32-bit values (a,b,c) into c

Functions

- uint32_t hashword (const uint32_t *k, size_t length, uint32_t initval)

  hash a variable-length key into a 32-bit value (Big Endian)
- void hashword2 (const uint32_t *k, size_t length, uint32_t *pc, uint32_t *pb)

  same as hashword(), but take two seeds and return two 32-bit values
- uint32_t hashlittle (const void *key, size_t length, uint32_t initval)

  hash a variable-length key into a 32-bit value (Little Endian)
- void hashlittle2 (const void *key, size_t length, uint32_t *pc, uint32_t *pb)

  return 2 32-bit hash values.
- uint32_t hashbig (const void *key, size_t length, uint32_t initval)

  This is the same as hashword() on big-endian machines.

14.30.1 Detailed Description

by Bob Jenkins, May 2006, Public Domain.

Definition in file lookup3.c.

14.31 lookup3.c

```
00001
00043 /*#define SELF_TEST 1*/
00044
00045 #include <stdio.h>      /* defines printf for tests */
00046 #include <time.h>       /* defines time_t for timings in the test */
00047 #include <stdint.h>     /* defines uint32_t etc */
00048 #include <sys/param.h>  /* attempt to define endianness */
00049 #ifdef linux
00050 # include <endian.h>    /* attempt to define endianness */
00051 #endif
00052
00053 /*
00054  * My best guess at if you are big-endian or little-endian.  This may
00055  * need adjustment.
00056  */
00057 #if (defined(__BYTE_ORDER) && defined(__LITTLE_ENDIAN) && \
00058      __BYTE_ORDER == __LITTLE_ENDIAN) || \
00059     (defined(i386) || defined(__i386__) || defined(__i486__) || \
00060      defined(__i586__) || defined(__i686__) || defined(vax) || defined(MIPSEL))
00061 # define HASH_LITTLE_ENDIAN 1
00062 # define HASH_BIG_ENDIAN 0
00063 #elif (defined(__BYTE_ORDER) && defined(__BIG_ENDIAN) && \
00064        __BYTE_ORDER == __BIG_ENDIAN) || \
00065       (defined(sparc) || defined(POWERPC) || defined(mc68000) || defined(sel))
00066 # define HASH_LITTLE_ENDIAN 0
00067 # define HASH_BIG_ENDIAN 1
00068 #else
00069 # define HASH_LITTLE_ENDIAN 0
00070 # define HASH_BIG_ENDIAN 0
00071 #endif
00072
00073 #define hashsize(n) ((uint32_t)1<<(n))
00074 #define hashmask(n) (hashsize(n)-1)
00075 #define rot(x,k) (((x)<<(k)) | ((x)>>(32-(k))))
00076
```

```
00077
00122 #define mix(a,b,c) \
00123 { \
00124   a -= c;  a ^= rot(c, 4);  c += b; \
00125   b -= a;  b ^= rot(a, 6);  a += c; \
00126   c -= b;  c ^= rot(b, 8);  b += a; \
00127   a -= c;  a ^= rot(c,16);  c += b; \
00128   b -= a;  b ^= rot(a,19);  a += c; \
00129   c -= b;  c ^= rot(b, 4);  b += a; \
00130 }
00131
00158 #define final(a,b,c) \
00159 { \
00160   c ^= b; c -= rot(b,14); \
00161   a ^= c; a -= rot(c,11); \
00162   b ^= a; b -= rot(a,25); \
00163   c ^= b; c -= rot(b,16); \
00164   a ^= c; a -= rot(c,4);  \
00165   b ^= a; b -= rot(a,14); \
00166   c ^= b; c -= rot(b,24); \
00167 }
00168
00182 uint32_t hashword(
00183     const uint32_t *k,                   /* the key, an array of uint32_t values */
00184     size_t          length,               /* the length of the key, in uint32_ts */
00185     uint32_t        initval) {         /* the previous hash, or an arbitrary value */
00186     uint32_t a,b,c;
00187
00188     /* Set up the internal state */
00189     a = b = c = 0xdeadbeef + (((uint32_t)length)<<2) + initval;
00190
00191     /*------------------------------------------------ handle most of the key */
00192     while (length > 3) {
00193         a += k[0];
00194         b += k[1];
00195         c += k[2];
00196         mix(a,b,c);
00197         length -= 3;
00198         k += 3;
00199     }
00200
00201     /*------------------------------------------- handle the last 3 uint32_t's */
00202     switch(length) {                     /* all the case statements fall through */
00203         case 3 :
00204             c+=k[2];
00205             /* no break */
00206         case 2 :
00207             b+=k[1];
00208             /* no break */
00209         case 1 :
00210             a+=k[0];
00211             final(a,b,c);
00212             /* no break */
00213         case 0:     /* case 0: nothing left to add */
00214             break;
00215     }
00216     /*------------------------------------------------------ report the result */
00217     return (c);
00218 }
00219
00220
00229 void hashword2 (
00230     const uint32_t *k,                   /* the key, an array of uint32_t values */
00231     size_t          length,               /* the length of the key, in uint32_ts */
00232     uint32_t       *pc,                      /* IN: seed OUT: primary hash value */
00233     uint32_t       *pb) {               /* IN: more seed OUT: secondary hash value */
00234     uint32_t a,b,c;
00235
00236     /* Set up the internal state */
00237     a = b = c = 0xdeadbeef + ((uint32_t)(length<<2)) + *pc;
00238     c += *pb;
00239
00240     /*------------------------------------------------ handle most of the key */
00241     while (length > 3) {
00242         a += k[0];
00243         b += k[1];
00244         c += k[2];
00245         mix(a,b,c);
00246         length -= 3;
00247         k += 3;
00248     }
00249
00250     /*------------------------------------------- handle the last 3 uint32_t's */
00251     switch(length) {                     /* all the case statements fall through */
00252         case 3 :
00253             c+=k[2];
00254             /* no break */
```

```
00255         case 2 :
00256             b+=k[1];
00257             /* no break */
00258         case 1 :
00259             a+=k[0];
00260             final(a,b,c);
00261             /* no break */
00262         case 0:     /* case 0: nothing left to add */
00263             break;
00264     }
00265     /*-------------------------------------------------------- report the result */
00266     *pc=c;
00267     *pb=b;
00268 }
00269
00270
00298 uint32_t hashlittle( const void *key, size_t length, uint32_t initval) {
00299     uint32_t a,b,c;                                          /* internal state */
00300     union {
00301         const void *ptr;
00302         size_t i;
00303     } u;     /* needed for Mac Powerbook G4 */
00304
00305     /* Set up the internal state */
00306     a = b = c = 0xdeadbeef + ((uint32_t)length) + initval;
00307
00308     u.ptr = key;
00309     if (HASH_LITTLE_ENDIAN && ((u.i & 0x3) == 0)) {
00310         const uint32_t *k = (const uint32_t *)key;         /* read 32-bit chunks */
00311 #ifdef VALGRIND
00312         const uint8_t  *k8;
00313 #endif
00314         /*------ all but last block: aligned reads and affect 32 bits of (a,b,c) */
00315         while (length > 12) {
00316             a += k[0];
00317             b += k[1];
00318             c += k[2];
00319             mix(a,b,c);
00320             length -= 12;
00321             k += 3;
00322         }
00323
00324         /*----------------------------- handle the last (probably partial) block */
00325         /*
00326          * "k[2]&0xffffff" actually reads beyond the end of the string, but
00327          * then masks off the part it's not allowed to read.  Because the
00328          * string is aligned, the masked-off tail is in the same word as the
00329          * rest of the string.  Every machine with memory protection I've seen
00330          * does it on word boundaries, so is OK with this.  But VALGRIND will
00331          * still catch it and complain.  The masking trick does make the hash
00332          * noticably faster for short strings (like English words).
00333          */
00334 #ifndef VALGRIND
00335
00336         switch(length) {
00337             case 12:
00338                 c+=k[2];
00339                 b+=k[1];
00340                 a+=k[0];
00341                 break;
00342             case 11:
00343                 c+=k[2]&0xffffff;
00344                 b+=k[1];
00345                 a+=k[0];
00346                 break;
00347             case 10:
00348                 c+=k[2]&0xffff;
00349                 b+=k[1];
00350                 a+=k[0];
00351                 break;
00352             case 9 :
00353                 c+=k[2]&0xff;
00354                 b+=k[1];
00355                 a+=k[0];
00356                 break;
00357             case 8 :
00358                 b+=k[1];
00359                 a+=k[0];
00360                 break;
00361             case 7 :
00362                 b+=k[1]&0xffffff;
00363                 a+=k[0];
00364                 break;
00365             case 6 :
00366                 b+=k[1]&0xffff;
00367                 a+=k[0];
00368                 break;
```

```
00369             case 5 :
00370                 b+=k[1]&0xff;
00371                 a+=k[0];
00372                 break;
00373             case 4 :
00374                 a+=k[0];
00375                 break;
00376             case 3 :
00377                 a+=k[0]&0xffffff;
00378                 break;
00379             case 2 :
00380                 a+=k[0]&0xffff;
00381                 break;
00382             case 1 :
00383                 a+=k[0]&0xff;
00384                 break;
00385             case 0 :
00386                 return (c);              /* zero length strings require no mixing */
00387         }
00388
00389 #else /* make valgrind happy */
00390
00391         k8 = (const uint8_t *)k;
00392         switch(length) {
00393             case 12:
00394                 c+=k[2];
00395                 b+=k[1];
00396                 a+=k[0];
00397                 break;
00398             case 11:
00399                 c+=((uint32_t)k8[10])<<16;  /* fall through */
00400             case 10:
00401                 c+=((uint32_t)k8[9])<<8;    /* fall through */
00402             case 9 :
00403                 c+=k8[8];                   /* fall through */
00404             case 8 :
00405                 b+=k[1];
00406                 a+=k[0];
00407                 break;
00408             case 7 :
00409                 b+=((uint32_t)k8[6])<<16;   /* fall through */
00410             case 6 :
00411                 b+=((uint32_t)k8[5])<<8;    /* fall through */
00412             case 5 :
00413                 b+=k8[4];                   /* fall through */
00414             case 4 :
00415                 a+=k[0];
00416                 break;
00417             case 3 :
00418                 a+=((uint32_t)k8[2])<<16;   /* fall through */
00419             case 2 :
00420                 a+=((uint32_t)k8[1])<<8;    /* fall through */
00421             case 1 :
00422                 a+=k8[0];
00423                 break;
00424             case 0 :
00425                 return c;
00426         }
00427
00428 #endif /* !valgrind */
00429
00430     } else
00431         if (HASH_LITTLE_ENDIAN && ((u.i & 0x1) == 0)) {
00432             const uint16_t *k = (const uint16_t *)key;         /* read 16-bit chunks */
00433             const uint8_t  *k8;
00434
00435             /*--------------- all but last block: aligned reads and different mixing */
00436             while (length > 12) {
00437                 a += k[0] + (((uint32_t)k[1])<<16);
00438                 b += k[2] + (((uint32_t)k[3])<<16);
00439                 c += k[4] + (((uint32_t)k[5])<<16);
00440                 mix(a,b,c);
00441                 length -= 12;
00442                 k += 6;
00443             }
00444
00445             /*----------------------------- handle the last (probably partial) block */
00446             k8 = (const uint8_t *)k;
00447             switch(length) {
00448                 case 12:
00449                     c+=k[4]+(((uint32_t)k[5])<<16);
00450                     b+=k[2]+(((uint32_t)k[3])<<16);
00451                     a+=k[0]+(((uint32_t)k[1])<<16);
00452                     break;
00453                 case 11:
00454                     c+=((uint32_t)k8[10])<<16;     /* fall through */
00455                     /* no break */
```

```
00456                 case 10:
00457                     c+=k[4];
00458                     b+=k[2]+(((uint32_t)k[3])<<16);
00459                     a+=k[0]+(((uint32_t)k[1])<<16);
00460                     break;
00461                 case 9 :
00462                     c+=k8[8];                      /* fall through */
00463                     /* no break */
00464                 case 8 :
00465                     b+=k[2]+(((uint32_t)k[3])<<16);
00466                     a+=k[0]+(((uint32_t)k[1])<<16);
00467                     break;
00468                 case 7 :
00469                     b+=((uint32_t)k8[6])<<16;      /* fall through */
00470                     /* no break */
00471                 case 6 :
00472                     b+=k[2];
00473                     a+=k[0]+(((uint32_t)k[1])<<16);
00474                     break;
00475                 case 5 :
00476                     b+=k8[4];                      /* fall through */
00477                     /* no break */
00478                 case 4 :
00479                     a+=k[0]+(((uint32_t)k[1])<<16);
00480                     break;
00481                 case 3 :
00482                     a+=((uint32_t)k8[2])<<16;      /* fall through */
00483                     /* no break */
00484                 case 2 :
00485                     a+=k[0];
00486                     break;
00487                 case 1 :
00488                     a+=k8[0];
00489                     break;
00490                 case 0 :
00491                     return (c);                     /* zero length requires no mixing */
00492             }
00493
00494         } else {                        /* need to read the key one byte at a time */
00495             const uint8_t *k = (const uint8_t *)key;
00496
00497             /*--------------- all but the last block: affect some 32 bits of (a,b,c) */
00498             while (length > 12) {
00499                 a += k[0];
00500                 a += ((uint32_t)k[1])<<8;
00501                 a += ((uint32_t)k[2])<<16;
00502                 a += ((uint32_t)k[3])<<24;
00503                 b += k[4];
00504                 b += ((uint32_t)k[5])<<8;
00505                 b += ((uint32_t)k[6])<<16;
00506                 b += ((uint32_t)k[7])<<24;
00507                 c += k[8];
00508                 c += ((uint32_t)k[9])<<8;
00509                 c += ((uint32_t)k[10])<<16;
00510                 c += ((uint32_t)k[11])<<24;
00511                 mix(a,b,c);
00512                 length -= 12;
00513                 k += 12;
00514             }
00515
00516             /*-------------------------------- last block: affect all 32 bits of (c) */
00517             switch(length) {                   /* all the case statements fall through */
00518                 case 12:
00519                     c+=((uint32_t)k[11])<<24;
00520                     /* no break */
00521                 case 11:
00522                     c+=((uint32_t)k[10])<<16;
00523                     /* no break */
00524                 case 10:
00525                     c+=((uint32_t)k[9])<<8;
00526                     /* no break */
00527                 case 9 :
00528                     c+=k[8];
00529                     /* no break */
00530                 case 8 :
00531                     b+=((uint32_t)k[7])<<24;
00532                     /* no break */
00533                 case 7 :
00534                     b+=((uint32_t)k[6])<<16;
00535                     /* no break */
00536                 case 6 :
00537                     b+=((uint32_t)k[5])<<8;
00538                     /* no break */
00539                 case 5 :
00540                     b+=k[4];
00541                     /* no break */
00542                 case 4 :
```

```
00543                     a+=((uint32_t)k[3])<<24;
00544                     /* no break */
00545                 case 3 :
00546                     a+=((uint32_t)k[2])<<16;
00547                     /* no break */
00548                 case 2 :
00549                     a+=((uint32_t)k[1])<<8;
00550                     /* no break */
00551                 case 1 :
00552                     a+=k[0];
00553                     break;
00554                 case 0 :
00555                     return (c);
00556             }
00557         }
00558 
00559     final(a,b,c);
00560     return (c);
00561 }
00562 
00563 
00574 void hashlittle2(
00575     const void *key,       /* the key to hash */
00576     size_t      length,    /* length of the key */
00577     uint32_t   *pc,        /* IN: primary initval, OUT: primary hash */
00578     uint32_t   *pb) {      /* IN: secondary initval, OUT: secondary hash */
00579     uint32_t a,b,c;                                          /* internal state */
00580     union {
00581         const void *ptr;
00582         size_t i;
00583     } u;     /* needed for Mac Powerbook G4 */
00584 
00585     /* Set up the internal state */
00586     a = b = c = 0xdeadbeef + ((uint32_t)length) + *pc;
00587     c += *pb;
00588 
00589     u.ptr = key;
00590     if (HASH_LITTLE_ENDIAN && ((u.i & 0x3) == 0)) {
00591         const uint32_t *k = (const uint32_t *)key;         /* read 32-bit chunks */
00592 #ifdef VALGRIND
00593         const uint8_t  *k8;
00594 #endif
00595 
00596         /*------ all but last block: aligned reads and affect 32 bits of (a,b,c) */
00597         while (length > 12) {
00598             a += k[0];
00599             b += k[1];
00600             c += k[2];
00601             mix(a,b,c);
00602             length -= 12;
00603             k += 3;
00604         }
00605 
00606         /*----------------------------- handle the last (probably partial) block */
00607         /*
00608          * "k[2]&0xffffff" actually reads beyond the end of the string, but
00609          * then masks off the part it's not allowed to read.  Because the
00610          * string is aligned, the masked-off tail is in the same word as the
00611          * rest of the string.  Every machine with memory protection I've seen
00612          * does it on word boundaries, so is OK with this.  But VALGRIND will
00613          * still catch it and complain.  The masking trick does make the hash
00614          * noticably faster for short strings (like English words).
00615          */
00616 #ifndef VALGRIND
00617 
00618         switch(length) {
00619             case 12:
00620                 c+=k[2];
00621                 b+=k[1];
00622                 a+=k[0];
00623                 break;
00624             case 11:
00625                 c+=k[2]&0xffffff;
00626                 b+=k[1];
00627                 a+=k[0];
00628                 break;
00629             case 10:
00630                 c+=k[2]&0xffff;
00631                 b+=k[1];
00632                 a+=k[0];
00633                 break;
00634             case 9 :
00635                 c+=k[2]&0xff;
00636                 b+=k[1];
00637                 a+=k[0];
00638                 break;
00639             case 8 :
```

```
00640                 b+=k[1];
00641                 a+=k[0];
00642                 break;
00643             case 7 :
00644                 b+=k[1]&0xffffff;
00645                 a+=k[0];
00646                 break;
00647             case 6 :
00648                 b+=k[1]&0xffff;
00649                 a+=k[0];
00650                 break;
00651             case 5 :
00652                 b+=k[1]&0xff;
00653                 a+=k[0];
00654                 break;
00655             case 4 :
00656                 a+=k[0];
00657                 break;
00658             case 3 :
00659                 a+=k[0]&0xffffff;
00660                 break;
00661             case 2 :
00662                 a+=k[0]&0xffff;
00663                 break;
00664             case 1 :
00665                 a+=k[0]&0xff;
00666                 break;
00667             case 0 :
00668                 *pc=c;
00669                 *pb=b;
00670                 return;  /* zero length strings require no mixing */
00671         }
00672
00673 #else /* make valgrind happy */
00674
00675         k8 = (const uint8_t *)k;
00676         switch(length) {
00677             case 12:
00678                 c+=k[2];
00679                 b+=k[1];
00680                 a+=k[0];
00681                 break;
00682             case 11:
00683                 c+=((uint32_t)k8[10])<<16;  /* fall through */
00684             case 10:
00685                 c+=((uint32_t)k8[9])<<8;    /* fall through */
00686             case 9 :
00687                 c+=k8[8];                   /* fall through */
00688             case 8 :
00689                 b+=k[1];
00690                 a+=k[0];
00691                 break;
00692             case 7 :
00693                 b+=((uint32_t)k8[6])<<16;   /* fall through */
00694             case 6 :
00695                 b+=((uint32_t)k8[5])<<8;    /* fall through */
00696             case 5 :
00697                 b+=k8[4];                   /* fall through */
00698             case 4 :
00699                 a+=k[0];
00700                 break;
00701             case 3 :
00702                 a+=((uint32_t)k8[2])<<16;   /* fall through */
00703             case 2 :
00704                 a+=((uint32_t)k8[1])<<8;    /* fall through */
00705             case 1 :
00706                 a+=k8[0];
00707                 break;
00708             case 0 :
00709                 *pc=c;
00710                 *pb=b;
00711                 return;  /* zero length strings require no mixing */
00712         }
00713
00714 #endif /* !valgrind */
00715
00716     } else
00717         if (HASH_LITTLE_ENDIAN && ((u.i & 0x1) == 0)) {
00718             const uint16_t *k = (const uint16_t *)key;         /* read 16-bit chunks */
00719             const uint8_t  *k8;
00720
00721             /*--------------- all but last block: aligned reads and different mixing */
00722             while (length > 12) {
00723                 a += k[0] + (((uint32_t)k[1])<<16);
00724                 b += k[2] + (((uint32_t)k[3])<<16);
00725                 c += k[4] + (((uint32_t)k[5])<<16);
00726                 mix(a,b,c);
```

```
00727                 length -= 12;
00728                 k += 6;
00729             }
00730 
00731             /*------------------------------ handle the last (probably partial) block */
00732             k8 = (const uint8_t *)k;
00733             switch(length) {
00734                 case 12:
00735                     c+=k[4]+(((uint32_t)k[5])<<16);
00736                     b+=k[2]+(((uint32_t)k[3])<<16);
00737                     a+=k[0]+(((uint32_t)k[1])<<16);
00738                     break;
00739                 case 11:
00740                     c+=((uint32_t)k8[10])<<16;     /* fall through */
00741                     /* no break */
00742                 case 10:
00743                     c+=k[4];
00744                     b+=k[2]+(((uint32_t)k[3])<<16);
00745                     a+=k[0]+(((uint32_t)k[1])<<16);
00746                     break;
00747                 case 9 :
00748                     c+=k8[8];                      /* fall through */
00749                     /* no break */
00750                 case 8 :
00751                     b+=k[2]+(((uint32_t)k[3])<<16);
00752                     a+=k[0]+(((uint32_t)k[1])<<16);
00753                     break;
00754                 case 7 :
00755                     b+=((uint32_t)k8[6])<<16;      /* fall through */
00756                     /* no break */
00757                 case 6 :
00758                     b+=k[2];
00759                     a+=k[0]+(((uint32_t)k[1])<<16);
00760                     break;
00761                 case 5 :
00762                     b+=k8[4];                      /* fall through */
00763                     /* no break */
00764                 case 4 :
00765                     a+=k[0]+(((uint32_t)k[1])<<16);
00766                     break;
00767                 case 3 :
00768                     a+=((uint32_t)k8[2])<<16;      /* fall through */
00769                     /* no break */
00770                 case 2 :
00771                     a+=k[0];
00772                     break;
00773                 case 1 :
00774                     a+=k8[0];
00775                     break;
00776                 case 0 :
00777                     *pc=c;
00778                     *pb=b;
00779                     return;  /* zero length strings require no mixing */
00780             }
00781 
00782         } else {                        /* need to read the key one byte at a time */
00783             const uint8_t *k = (const uint8_t *)key;
00784 
00785             /*--------------- all but the last block: affect some 32 bits of (a,b,c) */
00786             while (length > 12) {
00787                 a += k[0];
00788                 a += ((uint32_t)k[1])<<8;
00789                 a += ((uint32_t)k[2])<<16;
00790                 a += ((uint32_t)k[3])<<24;
00791                 b += k[4];
00792                 b += ((uint32_t)k[5])<<8;
00793                 b += ((uint32_t)k[6])<<16;
00794                 b += ((uint32_t)k[7])<<24;
00795                 c += k[8];
00796                 c += ((uint32_t)k[9])<<8;
00797                 c += ((uint32_t)k[10])<<16;
00798                 c += ((uint32_t)k[11])<<24;
00799                 mix(a,b,c);
00800                 length -= 12;
00801                 k += 12;
00802             }
00803 
00804             /*-------------------------------- last block: affect all 32 bits of (c) */
00805             switch(length) {                   /* all the case statements fall through */
00806                 case 12:
00807                     c+=((uint32_t)k[11])<<24;
00808                     /* no break */
00809                 case 11:
00810                     c+=((uint32_t)k[10])<<16;
00811                     /* no break */
00812                 case 10:
00813                     c+=((uint32_t)k[9])<<8;
```

```
00814                     /* no break */
00815                 case 9 :
00816                     c+=k[8];
00817                     /* no break */
00818                 case 8 :
00819                     b+=((uint32_t)k[7])<<24;
00820                     /* no break */
00821                 case 7 :
00822                     b+=((uint32_t)k[6])<<16;
00823                     /* no break */
00824                 case 6 :
00825                     b+=((uint32_t)k[5])<<8;
00826                     /* no break */
00827                 case 5 :
00828                     b+=k[4];
00829                     /* no break */
00830                 case 4 :
00831                     a+=((uint32_t)k[3])<<24;
00832                     /* no break */
00833                 case 3 :
00834                     a+=((uint32_t)k[2])<<16;
00835                     /* no break */
00836                 case 2 :
00837                     a+=((uint32_t)k[1])<<8;
00838                     /* no break */
00839                 case 1 :
00840                     a+=k[0];
00841                     break;
00842                 case 0 :
00843                     *pc=c;
00844                     *pb=b;
00845                     return;  /* zero length strings require no mixing */
00846             }
00847         }
00848
00849     final(a,b,c);
00850     *pc=c;
00851     *pb=b;
00852 }
00853
00854
00855
00862 uint32_t hashbig( const void *key, size_t length, uint32_t initval) {
00863     uint32_t a,b,c;
00864     union {
00865         const void *ptr;
00866         size_t i;
00867     } u; /* to cast key to (size_t) happily */
00868
00869     /* Set up the internal state */
00870     a = b = c = 0xdeadbeef + ((uint32_t)length) + initval;
00871
00872     u.ptr = key;
00873     if (HASH_BIG_ENDIAN && ((u.i & 0x3) == 0)) {
00874         const uint32_t *k = (const uint32_t *)key;         /* read 32-bit chunks */
00875 #ifdef VALGRIND
00876         const uint8_t  *k8;
00877 #endif
00878         /*------ all but last block: aligned reads and affect 32 bits of (a,b,c) */
00879         while (length > 12) {
00880             a += k[0];
00881             b += k[1];
00882             c += k[2];
00883             mix(a,b,c);
00884             length -= 12;
00885             k += 3;
00886         }
00887
00888         /*----------------------------- handle the last (probably partial) block */
00889         /*
00890          * "k[2]<<8" actually reads beyond the end of the string, but
00891          * then shifts out the part it's not allowed to read.  Because the
00892          * string is aligned, the illegal read is in the same word as the
00893          * rest of the string.  Every machine with memory protection I've seen
00894          * does it on word boundaries, so is OK with this.  But VALGRIND will
00895          * still catch it and complain.  The masking trick does make the hash
00896          * noticably faster for short strings (like English words).
00897          */
00898 #ifndef VALGRIND
00899
00900         switch(length) {
00901             case 12:
00902                 c+=k[2];
00903                 b+=k[1];
00904                 a+=k[0];
00905                 break;
00906             case 11:
```

```
00907                 c+=k[2]&0xffffff00;
00908                 b+=k[1];
00909                 a+=k[0];
00910                 break;
00911             case 10:
00912                 c+=k[2]&0xffff0000;
00913                 b+=k[1];
00914                 a+=k[0];
00915                 break;
00916             case 9 :
00917                 c+=k[2]&0xff000000;
00918                 b+=k[1];
00919                 a+=k[0];
00920                 break;
00921             case 8 :
00922                 b+=k[1];
00923                 a+=k[0];
00924                 break;
00925             case 7 :
00926                 b+=k[1]&0xffffff00;
00927                 a+=k[0];
00928                 break;
00929             case 6 :
00930                 b+=k[1]&0xffff0000;
00931                 a+=k[0];
00932                 break;
00933             case 5 :
00934                 b+=k[1]&0xff000000;
00935                 a+=k[0];
00936                 break;
00937             case 4 :
00938                 a+=k[0];
00939                 break;
00940             case 3 :
00941                 a+=k[0]&0xffffff00;
00942                 break;
00943             case 2 :
00944                 a+=k[0]&0xffff0000;
00945                 break;
00946             case 1 :
00947                 a+=k[0]&0xff000000;
00948                 break;
00949             case 0 :
00950                 return (c);              /* zero length strings require no mixing */
00951         }
00952
00953 #else  /* make valgrind happy */
00954
00955         k8 = (const uint8_t *)k;
00956         switch(length) {                 /* all the case statements fall through */
00957             case 12:
00958                 c+=k[2];
00959                 b+=k[1];
00960                 a+=k[0];
00961                 break;
00962             case 11:
00963                 c+=((uint32_t)k8[10])<<8;  /* fall through */
00964             case 10:
00965                 c+=((uint32_t)k8[9])<<16;  /* fall through */
00966             case 9 :
00967                 c+=((uint32_t)k8[8])<<24;  /* fall through */
00968             case 8 :
00969                 b+=k[1];
00970                 a+=k[0];
00971                 break;
00972             case 7 :
00973                 b+=((uint32_t)k8[6])<<8;   /* fall through */
00974             case 6 :
00975                 b+=((uint32_t)k8[5])<<16;  /* fall through */
00976             case 5 :
00977                 b+=((uint32_t)k8[4])<<24;  /* fall through */
00978             case 4 :
00979                 a+=k[0];
00980                 break;
00981             case 3 :
00982                 a+=((uint32_t)k8[2])<<8;   /* fall through */
00983             case 2 :
00984                 a+=((uint32_t)k8[1])<<16;  /* fall through */
00985             case 1 :
00986                 a+=((uint32_t)k8[0])<<24;
00987                 break;
00988             case 0 :
00989                 return c;
00990         }
00991
00992 #endif /* !VALGRIND */
00993
```

```
00994     } else {                        /* need to read the key one byte at a time */
00995         const uint8_t *k = (const uint8_t *)key;
00996 
00997         /*--------------- all but the last block: affect some 32 bits of (a,b,c) */
00998         while (length > 12) {
00999             a += ((uint32_t)k[0])<<24;
01000             a += ((uint32_t)k[1])<<16;
01001             a += ((uint32_t)k[2])<<8;
01002             a += ((uint32_t)k[3]);
01003             b += ((uint32_t)k[4])<<24;
01004             b += ((uint32_t)k[5])<<16;
01005             b += ((uint32_t)k[6])<<8;
01006             b += ((uint32_t)k[7]);
01007             c += ((uint32_t)k[8])<<24;
01008             c += ((uint32_t)k[9])<<16;
01009             c += ((uint32_t)k[10])<<8;
01010             c += ((uint32_t)k[11]);
01011             mix(a,b,c);
01012             length -= 12;
01013             k += 12;
01014         }
01015 
01016         /*-------------------------------- last block: affect all 32 bits of (c) */
01017         switch(length) {                   /* all the case statements fall through */
01018             case 12:
01019                 c+=k[11];
01020                 /* no break */
01021             case 11:
01022                 c+=((uint32_t)k[10])<<8;
01023                 /* no break */
01024             case 10:
01025                 c+=((uint32_t)k[9])<<16;
01026                 /* no break */
01027             case 9 :
01028                 c+=((uint32_t)k[8])<<24;
01029                 /* no break */
01030             case 8 :
01031                 b+=k[7];
01032                 /* no break */
01033             case 7 :
01034                 b+=((uint32_t)k[6])<<8;
01035                 /* no break */
01036             case 6 :
01037                 b+=((uint32_t)k[5])<<16;
01038                 /* no break */
01039             case 5 :
01040                 b+=((uint32_t)k[4])<<24;
01041                 /* no break */
01042             case 4 :
01043                 a+=k[3];
01044                 /* no break */
01045             case 3 :
01046                 a+=((uint32_t)k[2])<<8;
01047                 /* no break */
01048             case 2 :
01049                 a+=((uint32_t)k[1])<<16;
01050                 /* no break */
01051             case 1 :
01052                 a+=((uint32_t)k[0])<<24;
01053                 break;
01054             case 0 :
01055                 return (c);
01056         }
01057     }
01058 
01059     final(a,b,c);
01060     return (c);
01061 }
01062 
01065 #ifdef SELF_TEST
01066 
01067 /* used for timings */
01068 static void driver1() {
01069     uint8_t buf[256];
01070     uint32_t i;
01071     uint32_t h=0;
01072     time_t a,z;
01073 
01074     time(&a);
01075     for (i=0; i<256; ++i) {
01076         buf[i] = 'x';
01077     }
01078     for (i=0; i<1; ++i) {
01079         h = hashlittle(&buf[0],1,h);
01080     }
01081     time(&z);
01082     if (z-a > 0) {
```

```
01083         printf("time %d %.8x\n", z-a, h);
01084     }
01085 }
01086
01087 /* check that every input bit changes every output bit half the time */
01088 #define HASHSTATE 1
01089 #define HASHLEN   1
01090 #define MAXPAIR 60
01091 #define MAXLEN  70
01092 static void driver2() {
01093     uint8_t qa[MAXLEN+1], qb[MAXLEN+2], *a = &qa[0], *b = &qb[1];
01094     uint32_t c[HASHSTATE], d[HASHSTATE], i=0, j=0, k, l, m=0, z;
01095     uint32_t e[HASHSTATE],f[HASHSTATE],g[HASHSTATE],h[HASHSTATE];
01096     uint32_t x[HASHSTATE],y[HASHSTATE];
01097     uint32_t hlen;
01098
01099     printf("No more than %d trials should ever be needed \n",MAXPAIR/2);
01100     for (hlen=0; hlen < MAXLEN; ++hlen) {
01101         z=0;
01102         for (i=0; i<hlen; ++i) { /*------------------------ for each input byte, */
01103             for (j=0; j<8; ++j) { /*------------------------ for each input bit, */
01104                 for (m=1; m<8; ++m) { /*------------ for serveral possible initvals, */
01105                     for (l=0; l<HASHSTATE; ++l) {
01106                         e[l]=f[l]=g[l]=h[l]=x[l]=y[l]=~((uint32_t)0);
01107                     }
01108
01109                     /*---- check that every output bit is affected by that input bit */
01110                     for (k=0; k<MAXPAIR; k+=2) {
01111                         uint32_t finished=1;
01112                         /* keys have one bit different */
01113                         for (l=0; l<hlen+1; ++l) {
01114                             a[l] = b[l] = (uint8_t)0;
01115                         }
01116                         /* have a and b be two keys differing in only one bit */
01117                         a[i] ^= (k<<j);
01118                         a[i] ^= (k>>(8-j));
01119                         c[0] = hashlittle(a, hlen, m);
01120                         b[i] ^= ((k+1)<<j);
01121                         b[i] ^= ((k+1)>>(8-j));
01122                         d[0] = hashlittle(b, hlen, m);
01123                         /* check every bit is 1, 0, set, and not set at least once */
01124                         for (l=0; l<HASHSTATE; ++l) {
01125                             e[l] &= (c[l]^d[l]);
01126                             f[l] &= ~(c[l]^d[l]);
01127                             g[l] &= c[l];
01128                             h[l] &= ~c[l];
01129                             x[l] &= d[l];
01130                             y[l] &= ~d[l];
01131                             if (e[l]|f[l]|g[l]|h[l]|x[l]|y[l]) {
01132                                 finished=0;
01133                             }
01134                         }
01135                         if (finished) {
01136                             break;
01137                         }
01138                     }
01139                     if (k>z) {
01140                         z=k;
01141                     }
01142                     if (k==MAXPAIR) {
01143                         printf("Some bit didn't change: ");
01144                         printf("%.8x %.8x %.8x %.8x %.8x %.8x  ",
01145                                e[0],f[0],g[0],h[0],x[0],y[0]);
01146                         printf("i %d j %d m %d len %d\n", i, j, m, hlen);
01147                     }
01148                     if (z==MAXPAIR) {
01149                         goto done;
01150                     }
01151                 }
01152             }
01153         }
01154 done:
01155         if (z < MAXPAIR) {
01156             printf("Mix success  %2d bytes  %2d initvals  ",i,m);
01157             printf("required  %d  trials\n", z/2);
01158         }
01159     }
01160     printf("\n");
01161 }
01162
01163 /* Check for reading beyond the end of the buffer and alignment problems */
01164 static void driver3() {
01165     uint8_t buf[MAXLEN+20], *b;
01166     uint32_t len;
01167     uint8_t q[] = "This is the time for all good men to come to the aid of their country...";
01168     uint32_t h;
01169     uint8_t qq[] = "xThis is the time for all good men to come to the aid of their country...";
```

```
01170     uint32_t i;
01171     uint8_t qqq[] = "xxThis is the time for all good men to come to the aid of their country...";
01172     uint32_t j;
01173     uint8_t qqqq[] = "xxxThis is the time for all good men to come to the aid of their country...";
01174     uint32_t ref,x,y;
01175     uint8_t *p;
01176
01177     printf("Endianness.  These lines should all be the same (for values filled in):\n");
01178     printf("%.8x                            %.8x                            %.8x\n",
01179            hashword((const uint32_t *)q, (sizeof(q)-1)/4, 13),
01180            hashword((const uint32_t *)q, (sizeof(q)-5)/4, 13),
01181            hashword((const uint32_t *)q, (sizeof(q)-9)/4, 13));
01182     p = q;
01183     printf("%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x\n",
01184            hashlittle(p, sizeof(q)-1, 13), hashlittle(p, sizeof(q)-2, 13),
01185            hashlittle(p, sizeof(q)-3, 13), hashlittle(p, sizeof(q)-4, 13),
01186            hashlittle(p, sizeof(q)-5, 13), hashlittle(p, sizeof(q)-6, 13),
01187            hashlittle(p, sizeof(q)-7, 13), hashlittle(p, sizeof(q)-8, 13),
01188            hashlittle(p, sizeof(q)-9, 13), hashlittle(p, sizeof(q)-10, 13),
01189            hashlittle(p, sizeof(q)-11, 13), hashlittle(p, sizeof(q)-12, 13));
01190     p = &qq[1];
01191     printf("%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x\n",
01192            hashlittle(p, sizeof(q)-1, 13), hashlittle(p, sizeof(q)-2, 13),
01193            hashlittle(p, sizeof(q)-3, 13), hashlittle(p, sizeof(q)-4, 13),
01194            hashlittle(p, sizeof(q)-5, 13), hashlittle(p, sizeof(q)-6, 13),
01195            hashlittle(p, sizeof(q)-7, 13), hashlittle(p, sizeof(q)-8, 13),
01196            hashlittle(p, sizeof(q)-9, 13), hashlittle(p, sizeof(q)-10, 13),
01197            hashlittle(p, sizeof(q)-11, 13), hashlittle(p, sizeof(q)-12, 13));
01198     p = &qqq[2];
01199     printf("%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x\n",
01200            hashlittle(p, sizeof(q)-1, 13), hashlittle(p, sizeof(q)-2, 13),
01201            hashlittle(p, sizeof(q)-3, 13), hashlittle(p, sizeof(q)-4, 13),
01202            hashlittle(p, sizeof(q)-5, 13), hashlittle(p, sizeof(q)-6, 13),
01203            hashlittle(p, sizeof(q)-7, 13), hashlittle(p, sizeof(q)-8, 13),
01204            hashlittle(p, sizeof(q)-9, 13), hashlittle(p, sizeof(q)-10, 13),
01205            hashlittle(p, sizeof(q)-11, 13), hashlittle(p, sizeof(q)-12, 13));
01206     p = &qqqq[3];
01207     printf("%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x\n",
01208            hashlittle(p, sizeof(q)-1, 13), hashlittle(p, sizeof(q)-2, 13),
01209            hashlittle(p, sizeof(q)-3, 13), hashlittle(p, sizeof(q)-4, 13),
01210            hashlittle(p, sizeof(q)-5, 13), hashlittle(p, sizeof(q)-6, 13),
01211            hashlittle(p, sizeof(q)-7, 13), hashlittle(p, sizeof(q)-8, 13),
01212            hashlittle(p, sizeof(q)-9, 13), hashlittle(p, sizeof(q)-10, 13),
01213            hashlittle(p, sizeof(q)-11, 13), hashlittle(p, sizeof(q)-12, 13));
01214     printf("\n");
01215
01216     /* check that hashlittle2 and hashlittle produce the same results */
01217     i=47;
01218     j=0;
01219     hashlittle2(q, sizeof(q), &i, &j);
01220     if (hashlittle(q, sizeof(q), 47) != i) {
01221         printf("hashlittle2 and hashlittle mismatch\n");
01222     }
01223
01224     /* check that hashword2 and hashword produce the same results */
01225     len = 0xdeadbeef;
01226     i=47, j=0;
01227     hashword2(&len, 1, &i, &j);
01228     if (hashword(&len, 1, 47) != i)
01229         printf("hashword2 and hashword mismatch %x %x\n",
01230                i, hashword(&len, 1, 47));
01231
01232     /* check hashlittle doesn't read before or after the ends of the string */
01233     for (h=0, b=buf+1; h<8; ++h, ++b) {
01234         for (i=0; i<MAXLEN; ++i) {
01235             len = i;
01236             for (j=0; j<i; ++j) {
01237                 *(b+j)=0;
01238             }
01239
01240             /* these should all be equal */
01241             ref = hashlittle(b, len, (uint32_t)1);
01242             *(b+i)=(uint8_t)~0;
01243             *(b-1)=(uint8_t)~0;
01244             x = hashlittle(b, len, (uint32_t)1);
01245             y = hashlittle(b, len, (uint32_t)1);
01246             if ((ref != x) || (ref != y)) {
01247                 printf("alignment error: %.8x %.8x %.8x %d %d\n",ref,x,y,
01248                        h, i);
01249             }
01250         }
01251     }
01252 }
01253
01254 /* check for problems with nulls */
01255 static void driver4() {
01256     uint8_t buf[1];
```

```
01257     uint32_t h,i,state[HASHSTATE];
01258
01259
01260     buf[0] = ~0;
01261     for (i=0; i<HASHSTATE; ++i) {
01262         state[i] = 1;
01263     }
01264     printf("These should all be different\n");
01265     for (i=0, h=0; i<8; ++i) {
01266         h = hashlittle(buf, 0, h);
01267         printf("%2ld  0-byte strings, hash is  %.8x\n", i, h);
01268     }
01269 }
01270
01271 static void driver5() {
01272     uint32_t b,c;
01273     b=0, c=0, hashlittle2("", 0, &c, &b);
01274     printf("hash is %.8lx %.8lx\n", c, b);   /* deadbeef deadbeef */
01275     b=0xdeadbeef, c=0, hashlittle2("", 0, &c, &b);
01276     printf("hash is %.8lx %.8lx\n", c, b);   /* bd5b7dde deadbeef */
01277     b=0xdeadbeef, c=0xdeadbeef, hashlittle2("", 0, &c, &b);
01278     printf("hash is %.8lx %.8lx\n", c, b);   /* 9c093ccd bd5b7dde */
01279     b=0, c=0, hashlittle2("Four score and seven years ago", 30, &c, &b);
01280     printf("hash is %.8lx %.8lx\n", c, b);   /* 17770551 ce7226e6 */
01281     b=1, c=0, hashlittle2("Four score and seven years ago", 30, &c, &b);
01282     printf("hash is %.8lx %.8lx\n", c, b);   /* e3607cae bd371de4 */
01283     b=0, c=1, hashlittle2("Four score and seven years ago", 30, &c, &b);
01284     printf("hash is %.8lx %.8lx\n", c, b);   /* cd628161 6cbea4b3 */
01285     c = hashlittle("Four score and seven years ago", 30, 0);
01286     printf("hash is %.8lx\n", c);   /* 17770551 */
01287     c = hashlittle("Four score and seven years ago", 30, 1);
01288     printf("hash is %.8lx\n", c);   /* cd628161 */
01289 }
01290
01291
01292 static int main() {
01293     driver1();   /* test that the key is hashed: used for timings */
01294     driver2();   /* test that whole key is hashed thoroughly */
01295     driver3();   /* test that nothing but the key is hashed */
01296     driver4();   /* test hashing multiple buffers (all buffers are null) */
01297     driver5();   /* test the hash against known vectors */
01298     return 1;
01299 }
01300
01301 #endif  /* SELF_TEST */
```

14.32 src/main.c File Reference

Application framework.

```
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <sys/file.h>
#include "include/dtsapp.h"
#include "include/private.h"
```

Functions

- void printgnu (const char *pname, int year, const char *dev, const char *email, const char *www)

  Print a brief GNU copyright notice on console.

- void daemonize ()

  Daemonise the application using fork/exit.

- int lockpidfile (const char *runfile)

  Lock the run file in the framework application info.

- void framework_mkcore (char *progname, char *name, char *email, char *web, int year, char *runfile, int flags, syssighandler sigfunc)

  Initilise application data structure and return a reference.
- int framework_init (int argc, char *argv[], frameworkfunc callback)

  Initilise the application daemonise and join the manager thread.

14.32.1 Detailed Description

Application framework.

Definition in file main.c.

14.33 main.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00025 #include <unistd.h>
00026 #include <signal.h>
00027 #include <stdlib.h>
00028 #include <stdio.h>
00029 #include <stdint.h>
00030 #include <string.h>
00031 #include <fcntl.h>
00032 #include <sys/file.h>
00033
00034 #include "include/dtsapp.h"
00035 #include "include/private.h"
00036
00037 static struct framework_core *framework_core_info = NULL;
00038
00039 #ifndef __WIN32__
00040 /*
00041  * handle signals to cleanup gracefully on exit
00042  */
00043 static void framework_sig_handler(int sig, siginfo_t *si, void *unused) {
00044     /* flag and clean all threads*/
00045     switch (sig) {
00046         case SIGUSR1:
00047         case SIGUSR2:
00048         case SIGHUP:
00049         case SIGALRM:
00050             if (!thread_signal(sig) && framework_core_info->
      sig_handler) {
00051                 framework_core_info->sig_handler(sig, si, unused);
00052             }
00053             break;
00054         case SIGTERM:
00055         case SIGINT:
00056         default:
00057             if (!thread_signal(sig)) {
00058                 if (framework_core_info->sig_handler) {
00059                     framework_core_info->sig_handler(sig, si, unused);
00060                 } else {
00061                     stopthreads(1);
00062                     exit(-1);
00063                 }
00064             }
00065     }
00066 }
00067 #endif
00068
```

```
00078 extern void printgnu(const char *pname, int year, const char *dev, const char *
      email, const char *www) {
00079     printf("\n"
00080            "    %s\n\n"
00081            "    Copyright (C) %i %s <%s>\n\n"
00082            "        %s\n\n"
00083            "    This program comes with ABSOLUTELY NO WARRANTY\n"
00084            "    This is free software, and you are welcome to redistribute it\n"
00085            "    under certain conditions.\n\n", pname, year, dev, email, www);
00086 }
00087
00094 extern void daemonize() {
00095     struct framework_core *ci =  framework_core_info;
00096
00097 #ifndef __WIN32__
00098     pid_t   forkpid;
00099
00100     /* fork and die daemonize*/
00101     forkpid = fork();
00102     if (forkpid > 0) {
00103         /* im all grown up and can pass onto child*/
00104         exit(0);
00105     } else if (forkpid < 0) {
00106         /* could not fork*/
00107         exit(-1);
00108     }
00109
00110     setsid();
00111
00112     /* Dont want these as a daemon*/
00113     signal(SIGTSTP, SIG_IGN);
00114     signal(SIGCHLD, SIG_IGN);
00115 #endif
00116
00117     /*delayed lock file from FRAMEWORK_MAIN / framework_init*/
00118     if (ci && (ci->flags & FRAMEWORK_FLAG_DAEMONLOCK)) {
00119         if ((ci->flock = lockpidfile(ci->runfile)) < 0) {
00120             printf("Could not lock pid file Exiting\n");
00121             while(framework_core_info) {
00122                 objunref(framework_core_info);
00123             }
00124             exit (-1);
00125         }
00126         objunref(ci);
00127     }
00128 }
00129
00135 extern int lockpidfile(const char *runfile) {
00136     int lck_fd = 0;
00137 #ifndef __WIN32__
00138     char pidstr[12];
00139     pid_t   mypid;
00140
00141     mypid = getpid();
00142     sprintf(pidstr,"%i\n", (int)mypid);
00143     if (runfile && ((lck_fd = open(runfile, O_RDWR|O_CREAT, 0640)) > 0) && (!
      flock(lck_fd, LOCK_EX | LOCK_NB))) {
00144         if (write(lck_fd, pidstr, strlen(pidstr)) < 0) {
00145             close(lck_fd);
00146             lck_fd = -1;
00147         }
00148     /* file was opened and not locked*/
00149     } else if (runfile && lck_fd) {
00150         close(lck_fd);
00151         lck_fd = -1;
00152     }
00153 #endif
00154     return (lck_fd);
00155 }
00156
00157
00158 #ifndef __WIN32__
00159 /*
00160  * set up signal handler
00161  */
00162 static void configure_sigact(struct sigaction *sa) {
00163     sa->sa_flags = SA_SIGINFO | SA_RESTART;
00164     sigemptyset(&sa->sa_mask);
00165     sa->sa_sigaction = framework_sig_handler;
00166     sigaction(SIGINT, sa, NULL);
00167     sigaction(SIGTERM, sa, NULL);
00168
00169     /*internal interupts*/
00170     sigaction(SIGUSR1, sa, NULL);
00171     sigaction(SIGUSR2, sa, NULL);
00172     sigaction(SIGHUP, sa, NULL);
00173     sigaction(SIGALRM, sa, NULL);
```

```
00174 }
00175 #endif
00176
00177 /*
00178  * free core
00179  */
00180 static void framework_free(void *data) {
00181     struct framework_core *ci = data;
00182     framework_core_info = NULL;
00183
00184     if (ci->developer) {
00185         free((char *)ci->developer);
00186     }
00187     if (ci->email) {
00188         free((char *)ci->email);
00189     }
00190     if (ci->www) {
00191         free((char *)ci->www);
00192     }
00193     if (ci->sa) {
00194         free(ci->sa);
00195     }
00196     if (ci->flock >= 0) {
00197         close(ci->flock);
00198     }
00199     if (ci->runfile) {
00200         if (ci->flock >= 0) {
00201             unlink(ci->runfile);
00202         }
00203         free((char *)ci->runfile);
00204     }
00205 }
00206
00221 extern void framework_mkcore(char *progname, char *name, char *
      email, char *web, int year, char *runfile, int flags,
      syssighandler sigfunc) {
00222     struct framework_core *core_info;
00223     if (framework_core_info) {
00224         objunref(framework_core_info);
00225         framework_core_info = NULL;
00226     }
00227
00228     if (!(core_info = objalloc(sizeof(*core_info), framework_free))) {
00229         return;
00230     }
00231
00232 #ifndef __WIN32__
00233     if (core_info && !(core_info->sa = malloc(sizeof(*core_info->sa)))) {
00234         free(core_info);
00235         return;
00236     }
00237 #endif
00238
00239     ALLOC_CONST(core_info->developer, name);
00240     ALLOC_CONST(core_info->email, email);
00241     ALLOC_CONST(core_info->www, web);
00242     ALLOC_CONST(core_info->runfile, runfile);
00243     ALLOC_CONST(core_info->progname, progname);
00244     core_info->year = year;
00245     core_info->flags = flags;
00246 #ifndef __WIN32__
00247     core_info->sig_handler = sigfunc;
00248 #endif
00249     /* Pass reference to static system variable*/
00250     framework_core_info = core_info;
00251 }
00252
00260 extern int framework_init(int argc, char *argv[], frameworkfunc callback) {
00261     struct framework_core *ci =  framework_core_info;
00262     int ret = 0;
00263
00264     seedrand();
00265     sslstartup();
00266
00267     /*prinit out a GNU licence summary*/
00268     if (ci && !(ci->flags & FRAMEWORK_FLAG_NOGNU)) {
00269         printgnu(ci->progname, ci->year, ci->developer, ci->
      email, ci->www);
00270     }
00271
00272     /* grab a ref for framework_core_info to be used latter*/
00273     if (ci && ci->flags & FRAMEWORK_FLAG_DAEMONLOCK) {
00274         objref(ci);
00275     }
00276
00277     /* fork the process to daemonize it*/
00278     if (ci && ci->flags & FRAMEWORK_FLAG_DAEMON) {
```

```
00279         daemonize();
00280     }
00281
00282     /* write pid to lockfile this should be done post daemonize*/
00283     if (ci && !(ci->flags & FRAMEWORK_FLAG_DAEMONLOCK)) {
00284         if ((ci->flock = lockpidfile(ci->runfile)) < 0) {
00285             printf("Could not lock pid file Exiting\n");
00286             return -1;
00287         }
00288     }
00289
00290 #ifndef __WIN32__
00291     /* interupt handler close clean on term so physical is reset*/
00292     configure_sigact(framework_core_info->sa);
00293 #endif
00294
00295     /*run the code from the application*/
00296     if (callback) {
00297         ret = callback(argc, argv);
00298         /* wait for all threads to end*/
00299         stopthreads(1);
00300     }
00301
00302     /* turn off the lights*/
00303     objunref(ci);
00304     if (framework_core_info && framework_core_info->flags &
      FRAMEWORK_FLAG_DAEMONLOCK) {
00305         objunref(framework_core_info);
00306     }
00307     unrefconfigfiles();
00308     return (ret);
00309 }
00310
```

14.34 src/nf_ctrack.c File Reference

linux Netfilter Connection Tracking

```
#include "config.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <linux/types.h>
#include <linux/netfilter.h>
#include <libnetfilter_conntrack/libnetfilter_conntrack.h>
#include <libnetfilter_conntrack/libnetfilter_conntrack_tcp.h>
#include "include/dtsapp.h"
#include "include/private.h"
```

Data Structures

- struct **nfct_struct**

Enumerations

- enum NF_CTRACK_FLAGS { NFCTRACK_DONE = 1 << 0 }

  Netfilter Ctrack Flags.

Functions

- uint8_t nf_ctrack_init (void)
- struct nf_conntrack * nf_ctrack_buildct (uint8_t *pkt)
- uint8_t nf_ctrack_delete (uint8_t *pkt)
- uint8_t nf_ctrack_nat (uint8_t *pkt, uint32_t addr, uint16_t port, uint8_t dnat)
- void nf_ctrack_dump (void)
- struct nfct_struct * nf_ctrack_trace (void)
- void nf_ctrack_endtrace (struct nfct_struct *nfct)
- void nf_ctrack_close (void)

14.34.1 Detailed Description

linux Netfilter Connection Tracking

Definition in file nf_ctrack.c.

14.35 nf_ctrack.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00025 #include "config.h"
00026
00027 #include <stdint.h>
00028 #include <stdlib.h>
00029 #include <stdio.h>
00030 #include <unistd.h>
00031 #include <fcntl.h>
00032 #include <errno.h>
00033 #include <sys/ioctl.h>
00034 #include <netinet/in.h>
00035 #include <linux/types.h>
00036 #include <linux/netfilter.h>
00037 #include <libnetfilter_conntrack/libnetfilter_conntrack.h>
00038 #include <libnetfilter_conntrack/libnetfilter_conntrack_tcp.h>
00039
00040 #include "include/dtsapp.h"
00041 #include "include/private.h"
00042
00044 enum NF_CTRACK_FLAGS {
00045     NFCTRACK_DONE    = 1 << 0
00046 };
00047
00048 static struct nfct_struct {
00049     struct nfct_handle *nfct;
00050     int fd;
00051     int flags;
00052 } *ctrack = NULL;
00053
00054 static void close_nfct(void *data) {
00055     struct nfct_struct *nfct = data;
00056
00057     nfct_close(nfct->nfct);
00058 }
00059
00060 static int nfct_cb(enum nf_conntrack_msg_type type, struct nf_conntrack *ct, void *data) {
00061     char buf[1024];
00062
```

```
00063     nfct_snprintf(buf, sizeof(buf), ct, NFCT_T_UNKNOWN, NFCT_O_DEFAULT, NFCT_OF_SHOW_LAYER3 | NFCT_OF_TIME
      | NFCT_OF_TIMESTAMP);
00064     printf("%s\n", buf);
00065
00066     return (NFCT_CB_CONTINUE);
00067 }
00068
00069 static struct nfct_struct *nf_ctrack_alloc(uint8_t subsys_id, unsigned subscriptions) {
00070     struct nfct_struct *nfct;
00071
00072     if (!(nfct = objalloc((sizeof *ctrack), close_nfct))) {
00073         return (NULL);
00074     }
00075
00076     /* expectations and conntrack*/
00077     if (!(nfct->nfct = nfct_open(subsys_id, subscriptions))) {
00078         objunref(nfct);
00079         return (NULL);
00080     }
00081
00082     if ((nfct->fd = nfct_fd(nfct->nfct)) < 0) {
00083         objunref(nfct);
00084         return (NULL);
00085     }
00086
00087     return (nfct);
00088 }
00089
00090 extern uint8_t nf_ctrack_init(void) {
00091     if (!ctrack && !(ctrack = nf_ctrack_alloc(CONNTRACK, 0))) {
00092         return (-1);
00093     }
00094     return (0);
00095 }
00096
00097 extern struct nf_conntrack *nf_ctrack_buildct(uint8_t *pkt) {
00098     struct nf_conntrack *ct;
00099     struct iphdr *ip = (struct iphdr *)pkt;
00100     union l4hdr *l4 = (union l4hdr *)(pkt + (ip->ihl * 4));
00101
00102     if (!(ct = nfct_new())) {
00103         return (NULL);
00104     };
00105
00106     /*Build tuple*/
00107     nfct_set_attr_u8(ct, ATTR_L3PROTO, PF_INET);
00108     nfct_set_attr_u32(ct, ATTR_IPV4_SRC, ip->saddr);
00109     nfct_set_attr_u32(ct, ATTR_IPV4_DST, ip->daddr);
00110     nfct_set_attr_u8(ct, ATTR_L4PROTO, ip->protocol);
00111     switch(ip->protocol) {
00112         case IPPROTO_TCP:
00113             nfct_set_attr_u16(ct, ATTR_PORT_SRC, l4->tcp.source);
00114             nfct_set_attr_u16(ct, ATTR_PORT_DST, l4->tcp.dest);
00115             break;
00116         case IPPROTO_UDP:
00117             nfct_set_attr_u16(ct, ATTR_PORT_SRC, l4->udp.source);
00118             nfct_set_attr_u16(ct, ATTR_PORT_DST, l4->udp.dest);
00119             break;
00120         case IPPROTO_ICMP:
00121             nfct_set_attr_u8(ct, ATTR_ICMP_TYPE, l4->icmp.type);
00122             nfct_set_attr_u8(ct, ATTR_ICMP_CODE, l4->icmp.code);
00123             nfct_set_attr_u16(ct, ATTR_ICMP_ID, l4->icmp.un.echo.id);
00124             /* no break */
00125         default
00126                 :
00127             break;
00128     };
00129
00130     return (ct);
00131 }
00132
00133 extern uint8_t nf_ctrack_delete(uint8_t *pkt) {
00134     struct nf_conntrack *ct;
00135     uint8_t unref = 0;
00136     uint8_t ret = 0;
00137
00138     if (!ctrack) {
00139         if (nf_ctrack_init()) {
00140             return (-1);
00141         }
00142         unref = 1;
00143     }
00144
00145     ct = nf_ctrack_buildct(pkt);
00146     objlock(ctrack);
00147     if (nfct_query(ctrack->nfct, NFCT_Q_DESTROY, ct) < 0) {
00148         ret = -1;
```

```
00149     }
00150     objunlock(ctrack);
00151     nfct_destroy(ct);
00152
00153     if (unref) {
00154         nf_ctrack_close();
00155     }
00156
00157     return (ret);
00158 }
00159
00160 extern uint8_t nf_ctrack_nat(uint8_t *pkt, uint32_t addr, uint16_t port, uint8_t dnat) {
00161     struct iphdr *ip = (struct iphdr *)pkt;
00162     struct nf_conntrack *ct;
00163     uint8_t unref = 0;
00164     uint8_t ret = 0;
00165
00166     if (!ctrack) {
00167         if (nf_ctrack_init()) {
00168             return (-1);
00169         }
00170         unref = 1;
00171     }
00172
00173     ct = nf_ctrack_buildct(pkt);
00174     nfct_setobjopt(ct, NFCT_SOPT_SETUP_REPLY);
00175
00176     nfct_set_attr_u32(ct, ATTR_TIMEOUT, 120);
00177     nfct_set_attr_u32(ct, (dnat) ? ATTR_DNAT_IPV4 : ATTR_SNAT_IPV4, addr);
00178
00179     switch(ip->protocol) {
00180         case IPPROTO_TCP:
00181             nfct_set_attr_u8(ct, ATTR_TCP_STATE, TCP_CONNTRACK_ESTABLISHED);
00182             /* no break */
00183         case IPPROTO_UDP:
00184             if (port) {
00185                 nfct_set_attr_u16(ct, (dnat) ? ATTR_DNAT_PORT : ATTR_SNAT_PORT, port);
00186             }
00187             break;
00188     }
00189
00190     objlock(ctrack);
00191     if (nfct_query(ctrack->nfct, NFCT_Q_CREATE_UPDATE, ct) < 0) {
00192         ret = -1;
00193     }
00194     objunlock(ctrack);
00195     nfct_destroy(ct);
00196
00197     if (unref) {
00198         nf_ctrack_close();
00199     }
00200
00201     return (ret);
00202 }
00203
00204 extern void nf_ctrack_dump(void) {
00205     uint32_t family = PF_INET;
00206     uint8_t unref = 0;
00207
00208     if (!ctrack) {
00209         if (nf_ctrack_init()) {
00210             return;
00211         }
00212         unref = 1;
00213     }
00214
00215     objlock(ctrack);
00216     nfct_callback_register(ctrack->nfct, NFCT_T_ALL, nfct_cb, NULL);
00217     nfct_query(ctrack->nfct, NFCT_Q_DUMP, &family);
00218     nfct_callback_unregister(ctrack->nfct);
00219     objunlock(ctrack);
00220
00221     if (unref) {
00222         nf_ctrack_close();
00223     }
00224 }
00225
00226 static void *nf_ctrack_trace_th(void *data) {
00227     struct nfct_struct *nfct = data;
00228     fd_set  rd_set, act_set;
00229     struct timeval tv;
00230     int selfd;
00231     int opt = 1;
00232
00233     nfct_callback_register(nfct->nfct, NFCT_T_ALL, nfct_cb, NULL);
00234
00235     FD_ZERO(&rd_set);
```

```
00236     FD_SET(nfct->fd, &rd_set);
00237     fcntl(nfct->fd, F_SETFD, O_NONBLOCK);
00238     ioctl(nfct->fd, FIONBIO, &opt);
00239
00240     while (!testflag(nfct, NFCTRACK_DONE) &&
      framework_threadok()) {
00241         act_set = rd_set;
00242         tv.tv_sec = 0;
00243         tv.tv_usec = 20000;
00244         selfd = select(nfct->fd + 1, &act_set, NULL, NULL, &tv);
00245
00246         /*returned due to interupt continue or timed out*/
00247         if ((selfd < 0 && errno == EINTR) || (!selfd)) {
00248             continue;
00249         } else
00250             if (selfd < 0) {
00251                 break;
00252             }
00253
00254         if (FD_ISSET(nfct->fd, &act_set)) {
00255             nfct_catch(nfct->nfct);
00256         }
00257     }
00258     return (NULL);
00259 }
00260
00261 struct nfct_struct *nf_ctrack_trace(void) {
00262     struct nfct_struct *nfct;
00263     void *thr;
00264
00265     if (!(nfct = nf_ctrack_alloc(CONNTRACK, NFCT_ALL_CT_GROUPS))) {
00266         return (NULL);
00267     }
00268
00269     if (!(thr = framework_mkthread(nf_ctrack_trace_th, NULL, NULL, nfct,
      THREAD_OPTION_RETURN))) {
00270         objunref(nfct);
00271         return NULL;
00272     }
00273     objunref(thr);
00274
00275     return (nfct);
00276 }
00277
00278 extern void nf_ctrack_endtrace(struct nfct_struct *nfct) {
00279     if (nfct) {
00280         setflag(nfct, NFCTRACK_DONE);
00281     }
00282     objunref(nfct);
00283 }
00284
00285 extern void nf_ctrack_close(void) {
00286     if (ctrack) {
00287         objunref(ctrack);
00288     }
00289     ctrack = NULL;
00290 }
00291
```

14.36 src/nf_queue.c File Reference

Linux netfilter queue interface.

```
#include "config.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <linux/types.h>
#include <linux/netfilter.h>
#include <libnetfilter_queue/libnetfilter_queue.h>
#include "include/dtsapp.h"
#include "include/private.h"
```

Data Structures

- struct nfq_struct
- struct nfq_queue
- struct **nfq_list**

Enumerations

- enum NF_QUEUE_FLAGS { NFQUEUE_DONE = 1 << 0 }

Functions

- struct nfq_queue * nfqueue_attach (uint16_t pf, uint16_t num, uint8_t mode, uint32_t range, nfqueue_cb cb, void *data)
- uint16_t snprintf_pkt (struct nfq_data *tb, struct nfqnl_msg_packet_hdr *ph, uint8_t *pkt, char *buff, uint16_t len)

14.36.1 Detailed Description

Linux netfilter queue interface.

Definition in file nf_queue.c.

14.37 nf_queue.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
```

```
00018
00025 #include "config.h"
00026
00027 #include <stdint.h>
00028 #include <stdlib.h>
00029 #include <stdio.h>
00030 #include <string.h>
00031 #include <unistd.h>
00032 #include <fcntl.h>
00033 #include <errno.h>
00034 #include <sys/ioctl.h>
00035 #include <netinet/in.h>
00036 #include <linux/types.h>
00037 #include <linux/netfilter.h>
00038 #include <libnetfilter_queue/libnetfilter_queue.h>
00039
00040 #include "include/dtsapp.h"
00041 #include "include/private.h"
00042
00043 enum NF_QUEUE_FLAGS {
00044     NFQUEUE_DONE    = 1 << 0
00045 };
00046
00047 struct nfq_struct {
00048     struct nfq_handle *h;
00049     uint16_t pf;
00050     int fd;
00051     int flags;
00052 };
00053
00054 struct nfq_queue {
00055     struct nfq_struct *nfq;
00056     struct nfq_q_handle *qh;
00057     nfqueue_cb cb;
00058     void *data;
00059     uint16_t num;
00060 };
00061
00062 static struct nfq_list {
00063     struct bucket_list *queues;
00064 }  *nfqueues = NULL;
00065
00066 static int32_t nfqueue_hash(const void *data, int key) {
00067     const struct nfq_struct *nfq = data;
00068     const uint16_t *hashkey = (key) ? data : &nfq->pf;
00069
00070     return (*hashkey);
00071 }
00072
00073 static void nfqueues_close(void *data) {
00074
00075     if (nfqueues->queues) {
00076         objunref(nfqueues->queues);
00077     }
00078     nfqueues = NULL;
00079 }
00080
00081 static void nfqueue_close(void *data) {
00082     struct nfq_struct *nfq = data;
00083
00084     nfq_unbind_pf(nfq->h, nfq->pf);
00085     nfq_close(nfq->h);
00086     objunref(nfqueues);
00087 }
00088
00089 static void nfqueue_close_q(void *data) {
00090     struct nfq_queue *nfq_q = data;
00091
00092     if (nfq_q->qh) {
00093         nfq_destroy_queue(nfq_q->qh);
00094     }
00095
00096     /*im here in the list and running thread*/
00097     objlock(nfqueues);
00098     if (objcnt(nfq_q->nfq) <= 3) {
00099         setflag(nfq_q->nfq, NFQUEUE_DONE);
00100         remove_bucket_item(nfqueues->queues, nfq_q->nfq);
00101     }
00102     objunlock(nfqueues);
00103     objunref(nfq_q->nfq);
00104 }
00105
00106 static void *nfqueue_thread(void *data) {
00107     struct nfq_struct *nfq = data;
00108     fd_set  rd_set, act_set;
00109     struct timeval tv;
00110     int len, selfd;
```

```
00111     char buf[4096];
00112     int opt = 1;
00113
00114     FD_ZERO(&rd_set);
00115     FD_SET(nfq->fd, &rd_set);
00116     fcntl(nfq->fd, F_SETFD, O_NONBLOCK);
00117     ioctl(nfq->fd, FIONBIO, &opt);
00118
00119     while (!testflag(nfq, NFQUEUE_DONE) &&
      framework_threadok()) {
00120         act_set = rd_set;
00121         tv.tv_sec = 0;
00122         tv.tv_usec = 20000;
00123
00124         selfd = select(nfq->fd + 1, &act_set, NULL, NULL, &tv);
00125
00126         /*returned due to interupt continue or timed out*/
00127         if ((selfd < 0 && errno == EINTR) || (!selfd)) {
00128             continue;
00129         } else
00130             if (selfd < 0) {
00131                 break;
00132             }
00133
00134         if ((FD_ISSET(nfq->fd, &act_set)) &&
00135                 ((len = recv(nfq->fd, buf, sizeof(buf), 0)) >= 0)) {
00136             objlock(nfq);
00137             nfq_handle_packet(nfq->h, buf, len);
00138             objunlock(nfq);
00139         }
00140     }
00141
00142     return (NULL);
00143 }
00144
00145 static struct nfq_struct *nfqueue_init(uint16_t pf) {
00146     struct nfq_struct *nfq;
00147
00148     if (!(nfq = objalloc(sizeof(*nfq), nfqueue_close))) {
00149         return (NULL);
00150     }
00151     nfq->pf = pf;
00152
00153     if (!(nfq->h = nfq_open())) {
00154         objunref(nfq);
00155         return (NULL);
00156     }
00157
00158     if (nfq_unbind_pf(nfq->h, pf)) {
00159         objunref(nfq);
00160         return (NULL);
00161     }
00162
00163     if (nfq_bind_pf(nfq->h, pf)) {
00164         objunref(nfq);
00165         return (NULL);
00166     }
00167
00168     if ((nfq->fd = nfq_fd(nfq->h)) < 0) {
00169         objunref(nfq);
00170         return (NULL);
00171     }
00172
00173     if (nfqueues) {
00174         objref(nfqueues);
00175     } else
00176         if (!(nfqueues = objalloc(sizeof(*nfqueues), nfqueues_close))) {
00177             objunref(nfq);
00178             return (NULL);
00179         }
00180
00181     objlock(nfqueues);
00182     if ((nfqueues->queues || (nfqueues->queues = create_bucketlist(0, nfqueue_hash))) &&
00183             !addtobucket(nfqueues->queues, nfq)) {
00184         objunref(nfqueues);
00185         objunref(nfq);
00186         return (NULL);
00187     }
00188     objunlock(nfqueues);
00189
00190     framework_mkthread(nfqueue_thread, NULL, NULL, nfq, 0);
00191
00192     return (nfq);
00193 }
00194
00195 static int nfqueue_callback(struct nfq_q_handle *qh, struct nfgenmsg *msg, struct
      nfq_data *nfad, void *data) {
```

```
00196     struct nfq_queue *nfq_q = data;
00197     unsigned char *pkt;
00198     struct nfqnl_msg_packet_hdr *ph;
00199     void *mangle = NULL;
00200     uint32_t ret, mark;
00201     uint32_t id = 0;
00202     uint32_t len = 0;
00203     uint32_t verdict = NF_DROP;
00204
00205     if ((ph = nfq_get_msg_packet_hdr(nfad))) {
00206         id = ntohl(ph->packet_id);
00207     }
00208     mark = nfq_get_nfmark(nfad);
00209
00210     if ((len = nfq_get_payload(nfad, &pkt)) <= 0) {
00211         pkt = NULL;
00212     }
00213
00214     if (nfq_q->cb) {
00215         verdict = nfq_q->cb(nfad, ph, (char *)pkt, len, nfq_q->data, &mark, &mangle);
00216     }
00217
00218     if (mangle && !(len = objsize(mangle))) {
00219         objunref(mangle);
00220         mangle = NULL;
00221     }
00222
00223     ret = nfq_set_verdict2(qh, id, verdict, mark, len, (mangle) ? mangle : pkt);
00224     if (mangle) {
00225         objunref(mangle);
00226     }
00227
00228     return (ret);
00229 }
00230
00231 extern struct nfq_queue *nfqueue_attach(uint16_t pf, uint16_t
      num, uint8_t mode, uint32_t range, nfqueue_cb cb, void *data) {
00232     struct nfq_queue *nfq_q;
00233
00234     if (!(nfq_q = objalloc(sizeof(*nfq_q), nfqueue_close_q))) {
00235         return (NULL);
00236     }
00237
00238     objlock(nfqueues);
00239     if (!(nfqueues && (nfq_q->nfq = bucket_list_find_key(nfqueues->queues, &pf))) &&
00240             !(nfq_q->nfq || (nfq_q->nfq = nfqueue_init(pf)))) {
00241         objunlock(nfqueues);
00242         objunref(nfq_q);
00243         return (NULL);
00244     }
00245     objunlock(nfqueues);
00246
00247     if (!(nfq_q->qh = nfq_create_queue(nfq_q->nfq->h, num, &nfqueue_callback, nfq_q))) {
00248         objunref(nfq_q);
00249         return (NULL);
00250     }
00251
00252     if (cb) {
00253         nfq_q->cb = cb;
00254     }
00255
00256     if (data) {
00257         nfq_q->data = data;
00258     }
00259
00260     nfq_set_mode(nfq_q->qh, mode, range);
00261
00262     return (nfq_q);
00263 }
00264
00265 extern uint16_t snprintf_pkt(struct nfq_data *tb, struct
      nfqnl_msg_packet_hdr *ph, uint8_t *pkt, char *buff, uint16_t len) {
00266     struct iphdr *ip = (struct iphdr *)pkt;
00267     char *tmp = buff;
00268     uint32_t id, mark, ifi;
00269     uint16_t tlen, left = len;
00270     char saddr[INET_ADDRSTRLEN], daddr[INET_ADDRSTRLEN];
00271
00272     if (ph) {
00273         id = ntohl(ph->packet_id);
00274         snprintf(tmp, left, "hw_protocol=0x%04x hook=%u id=%u ",
00275                 ntohs(ph->hw_protocol), ph->hook, id);
00276         tlen = strlen(tmp);
00277         tmp += tlen;
00278         left -= tlen;
00279     }
00280
```

```
00281     if ((mark = nfq_get_nfmark(tb))) {
00282         snprintf(tmp, left, "mark=%u ", mark);
00283         tlen = strlen(tmp);
00284         tmp += tlen;
00285         left -= tlen;
00286     }
00287
00288     if ((ifi = nfq_get_indev(tb))) {
00289         snprintf(tmp, left, "indev=%u ", ifi);
00290         tlen = strlen(tmp);
00291         tmp += tlen;
00292         left -= tlen;
00293     }
00294
00295     if ((ifi = nfq_get_outdev(tb))) {
00296         snprintf(tmp, left, "outdev=%u ", ifi);
00297         tlen = strlen(tmp);
00298         tmp += tlen;
00299         left -= tlen;
00300     }
00301
00302     if (pkt && (ip->version == 4)) {
00303         union l4hdr *l4 = (union l4hdr *)(pkt + (ip->ihl*4));
00304
00305         inet_ntop(AF_INET, &ip->saddr, saddr, INET_ADDRSTRLEN);
00306         inet_ntop(AF_INET, &ip->daddr, daddr, INET_ADDRSTRLEN);
00307
00308         snprintf(tmp, left, "src=%s dst=%s proto=%i ", saddr, daddr, ip->protocol);
00309         tlen = strlen(tmp);
00310         tmp += tlen;
00311         left -= tlen;
00312
00313         switch(ip->protocol) {
00314             case IPPROTO_TCP:
00315                 snprintf(tmp, left, "sport=%i dport=%i ", ntohs(l4->tcp.source), ntohs(l4->tcp.dest));
00316                 break;
00317             case IPPROTO_UDP:
00318                 snprintf(tmp, left, "sport=%i dport=%i ", ntohs(l4->udp.source), ntohs(l4->udp.dest));
00319                 break;
00320             case IPPROTO_ICMP:
00321                 snprintf(tmp, left, "type=%i code=%i id=%i ", l4->icmp.type, l4->icmp.code, ntohs(l4->icmp.
      un.echo.id));
00322                 break;
00323         }
00324         tlen = strlen(tmp);
00325         tmp += tlen;
00326         left -= tlen;
00327     }
00328
00329     return (len - left);
00330 }
00331
```

14.38 src/openldap.c File Reference

Openldap/SASL Implementation.

```
#include <ldap.h>
#include <ldap_schema.h>
#include <lber.h>
#include <sasl/sasl.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <ctype.h>
#include <sys/time.h>
#include <stdarg.h>
#include "include/dtsapp.h"
```

Data Structures

- struct sasl_defaults

SASL Paramaters used in authentification.

- struct ldap_simple

  LDAP Simple bind.
- struct ldap_conn

  LDAP connection.
- struct ldap_modify

  LDAP Modify structure.
- struct ldap_add

  LDAP Add structure.
- struct ldap_modval

  Linked list of mod values.
- struct ldap_modreq

  LDAP mod request.

Functions

- struct ldap_conn ∗ ldap_connect (const char ∗uri, enum ldap_starttls starttls, int timelimit, int limit, int debug, int ∗err)

  Connect to a LDAP server.
- int ldap_simplebind (struct ldap_conn ∗ld, const char ∗dn, const char ∗passwd)

  Bind to the connection with simple bind requireing a distingushed name and password.
- int ldap_simplerebind (struct ldap_conn ∗ldap, const char ∗initialdn, const char ∗initialpw, const char ∗base, const char ∗filter, const char ∗uidrdn, const char ∗uid, const char ∗passwd)

  Bind to LDAP connection using rebind.
- int ldap_saslbind (struct ldap_conn ∗ld, const char ∗mech, const char ∗realm, const char ∗authcid, const char ∗passwd, const char ∗authzid)

  Bind to the server with SASL.
- const char ∗ ldap_errmsg (int res)

  Return LDAP error for a ldap error.
- struct ldap_results ∗ ldap_search_sub (struct ldap_conn ∗ld, const char ∗base, const char ∗filter, int b64enc, int ∗res,...)

  Search LDAP connection subtree.
- struct ldap_results ∗ ldap_search_one (struct ldap_conn ∗ld, const char ∗base, const char ∗filter, int b64enc, int ∗res,...)

  Search LDAP connection one level.
- struct ldap_results ∗ ldap_search_base (struct ldap_conn ∗ld, const char ∗base, const char ∗filter, int b64enc, int ∗res,...)

  Search LDAP connection base.
- void ldap_unref_attr (struct ldap_entry ∗entry, struct ldap_attr ∗attr)

  Remove a attribute from a entry.
- void ldap_unref_entry (struct ldap_results ∗results, struct ldap_entry ∗entry)

  Remove a entry from a result.
- struct ldap_entry ∗ ldap_getentry (struct ldap_results ∗results, const char ∗dn)

  Find and return the entry from the results for a specific dn.
- struct ldap_attr ∗ ldap_getattr (struct ldap_entry ∗entry, const char ∗attr)

  Find and return attribute in a entry.
- struct ldap_modify ∗ ldap_modifyinit (const char ∗dn)

  Create a modification reference for a DN.
- int ldap_mod_del (struct ldap_modify ∗lmod, const char ∗attr,...)

  Delete values from a attribute.
- int ldap_mod_add (struct ldap_modify ∗lmod, const char ∗attr,...)

Add values to a attribute.

- int ldap_mod_rep (struct ldap_modify *lmod, const char *attr,...)

  Replace a attribute.

- int ldap_domodify (struct ldap_conn *ld, struct ldap_modify *lmod)

  Apply the modification to the server.

- int ldap_mod_delattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Delete a value from a attribute in a DN.

- int ldap_mod_rematttr (struct ldap_conn *ldap, const char *dn, const char *attr)

  Delete a attribute from a DN.

- int ldap_mod_addattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Add a value for a attribute in a DN.

- int ldap_mod_repattr (struct ldap_conn *ldap, const char *dn, const char *attr, const char *value)

  Replace the value of a attribute in a DN.

- struct ldap_add * ldap_addinit (const char *dn)

  Create a reference to add a new DN.

- int ldap_add_attr (struct ldap_add *ladd, const char *attr,...)

  Add a attribute to new DN.

- int ldap_doadd (struct ldap_conn *ld, struct ldap_add *ladd)

  Write new DN to server.

14.38.1 Detailed Description

Openldap/SASL Implementation.

Definition in file openldap.c.

14.39 openldap.c

```
00001 #include <ldap.h>
00002 #include <ldap_schema.h>
00003 #include <lber.h>
00004 #include <sasl/sasl.h>
00005 #include <stdlib.h>
00006 #include <stdint.h>
00007 #include <stdio.h>
00008 #include <ctype.h>
00009 #include <sys/time.h>
00010 #include <stdarg.h>
00011
00012 #include "include/dtsapp.h"
00013
00014 static struct ldap_results *_dtsldapsearch(struct ldap_conn *ldap, const char *base,
      int scope,
00015                     const char *filter, char **attrs, int b64enc, int *err);
00016
00023 /*
00024  * http://www.opensource.apple.com/source/OpenLDAP/OpenLDAP-186/OpenLDAP/libraries/liblutil/sasl.c
00025  */
00026
00028 struct sasl_defaults {
00030     const char *mech;
00032     const char *realm;
00034     const char *authcid;
00036     const char *passwd;
00038     const char *authzid;
00039 };
00040
00042 struct ldap_simple {
00044     const char *dn;
00046     struct berval *cred;
00047 };
00048
00050 struct ldap_conn {
00052     LDAP    *ldap;
00054     char    *uri;
00056     int timelim;
```

```
00058     int limit;
00060     LDAPControl **sctrlsp;
00062     struct sasl_defaults *sasl;
00064     struct ldap_simple *simple;
00065 };
00066
00068 struct ldap_modify {
00070     const char *dn;
00072     struct bucket_list *bl[3];
00073 };
00074
00076 struct ldap_add {
00078     const char *dn;
00080     struct bucket_list *bl;
00081 };
00082
00084 struct ldap_modval {
00086     const char *value;
00088     struct ldap_modval *next;
00089 };
00090
00092 struct ldap_modreq {
00094     const char *attr;
00096     int cnt;
00098     struct ldap_modval *first;
00100     struct ldap_modval *last;
00101 };
00102
00103 static int ldap_count(LDAP *ld, LDAPMessage *message, int *err);
00104 static struct ldap_entry *ldap_getent(LDAP *ld, LDAPMessage **msgptr, LDAPMessage *result, int
      b64enc, int *err);
00105 static int dts_sasl_interact(LDAP *ld, unsigned flags, void *defaults, void *in );
00106
00107 static void free_simple(void *data) {
00108     struct ldap_simple *simple = data;
00109     struct berval *bv = simple->cred;
00110
00111     if (bv && bv->bv_val) {
00112         free(bv->bv_val);
00113     }
00114     if (bv) {
00115         free(bv);
00116     }
00117     if (simple->dn) {
00118         free((void *)simple->dn);
00119     }
00120 }
00121
00122 static void free_modval(void *data) {
00123     struct ldap_modval *modv = data;
00124
00125     if (modv->value) {
00126         free((void *)modv->value);
00127     }
00128 }
00129
00130 static void free_modreq(void *data) {
00131     struct ldap_modreq *modr = data;
00132     struct ldap_modval *modv;
00133
00134     if (modr->attr) {
00135         free((void *)modr->attr);
00136     }
00137     for(modv = modr->first; modv; modv = modv->next) {
00138         objunref(modv);
00139     }
00140 }
00141
00142 static void free_modify(void *data) {
00143     struct ldap_modify *lmod = data;
00144     int cnt;
00145     if (lmod->dn) {
00146         free((void *)lmod->dn);
00147     }
00148
00149     for(cnt=0; cnt < 3; cnt++) {
00150         if (lmod->bl[cnt]) {
00151             objunref(lmod->bl[cnt]);
00152         }
00153     }
00154 }
00155
00156 static void free_add(void *data) {
00157     struct ldap_add *lmod = data;
00158
00159     if (lmod->dn) {
00160         free((void *)lmod->dn);
```

```
00161     }
00162
00163     if (lmod->bl) {
00164         objunref(lmod->bl);
00165     }
00166 }
00167
00168 static void free_sasl(void *data) {
00169     struct sasl_defaults *sasl = data;
00170
00171     if (sasl->mech) {
00172         free((void *)sasl->mech);
00173     }
00174     if (sasl->realm) {
00175         free((void *)sasl->realm);
00176     }
00177     if (sasl->authcid) {
00178         free((void *)sasl->authcid);
00179     }
00180     if (sasl->passwd) {
00181         free((void *)sasl->passwd);
00182     }
00183     if (sasl->authzid) {
00184         free((void *)sasl->authzid);
00185     }
00186 }
00187
00188 static void free_ldapconn(void *data) {
00189     struct ldap_conn *ld = data;
00190
00191
00192     if (ld->uri) {
00193         free(ld->uri);
00194     }
00195     if (ld->ldap) {
00196         ldap_unbind_ext_s(ld->ldap, ld->sctrlsp, NULL);
00197     }
00198     if (ld->sasl) {
00199         objunref(ld->sasl);
00200     }
00201     if (ld->simple) {
00202         objunref(ld->simple);
00203     }
00204 }
00205
00206 static void free_result(void *data) {
00207     struct ldap_results *res = data;
00208     if (res->entries) {
00209         objunref(res->entries);
00210     }
00211 }
00212
00213 static void free_entry(void *data) {
00214     struct ldap_entry *ent = data;
00215     struct ldap_attr *la;
00216
00217     if (ent->prev) {
00218         ent->prev->next = ent->next;
00219     }
00220     if (ent->next) {
00221         ent->next->prev = ent->prev;
00222     }
00223
00224     if (ent->dn) {
00225         ldap_memfree((void *)ent->dn);
00226     }
00227     if (ent->rdn) {
00228         objunref(ent->rdn);
00229     }
00230     if (ent->dnufn) {
00231         free((void *)ent->dnufn);
00232     }
00233     if (ent->attrs) {
00234         objunref(ent->attrs);
00235     }
00236     if (ent->first_attr) {
00237         for(la = ent->first_attr; la; la = la->next) {
00238             objunref(la);
00239         }
00240     }
00241 }
00242
00243 static void free_rdnarr(void *data) {
00244     struct ldap_rdn **rdn = data;
00245
00246     for(; *rdn; rdn++) {
00247         objunref(*rdn);
```

```
00248     }
00249 }
00250
00251 static void free_rdn(void *data) {
00252     struct ldap_rdn *rdn = data;
00253
00254     if (rdn->name) {
00255         objunref((void *)rdn->name);
00256     }
00257     if (rdn->value) {
00258         objunref((void *)rdn->value);
00259     }
00260 }
00261
00262 static void free_attr(void *data) {
00263     struct ldap_attr *la = data;
00264     if (la->next) {
00265         la->next->prev = la->prev;
00266     }
00267     if (la->prev) {
00268         la->prev->next = la->next;
00269     }
00270     ldap_memfree((char *)la->name);
00271     if (la->vals) {
00272         objunref(la->vals);
00273     }
00274 }
00275
00276 static void free_attrvalarr(void *data) {
00277     struct ldap_attrval **av = data;
00278     for(; *av; av++) {
00279         objunref(*av);
00280     }
00281 }
00282
00283 static void free_attrval(void *data) {
00284     struct ldap_attrval *av = data;
00285     if (av->buffer) {
00286         objunref(av->buffer);
00287     }
00288 }
00289
00290 static int32_t modify_hash(const void *data, int key) {
00291     int ret;
00292     const struct ldap_modreq *modr = data;
00293     const char *hashkey = (key) ? data : modr->attr;
00294
00295     if (hashkey) {
00296         ret = jenhash(hashkey, strlen(hashkey), 0);
00297     } else {
00298         ret = jenhash(modr, sizeof(modr), 0);
00299     }
00300     return(ret);
00301 }
00302
00303 static int ldap_rebind_proc(LDAP *ld, LDAP_CONST char *url, ber_tag_t request, ber_int_t msgid, void *
      params) {
00304     struct ldap_conn *ldap = params;
00305     int res = LDAP_UNAVAILABLE;
00306
00307     if (!objref(ldap)) {
00308         return LDAP_UNAVAILABLE;
00309     }
00310
00311     if (ldap->sasl) {
00312         int sasl_flags = LDAP_SASL_AUTOMATIC | LDAP_SASL_QUIET;
00313         struct sasl_defaults *sasl = ldap->sasl;
00314
00315         res = ldap_sasl_interactive_bind_s(ld, NULL, sasl->mech, ldap->
      sctrlsp , NULL, sasl_flags, dts_sasl_interact, sasl);
00316     } else
00317         if (ldap->simple) {
00318             struct ldap_simple *simple = ldap->simple;
00319
00320             res = ldap_sasl_bind_s(ld, simple->dn, LDAP_SASL_SIMPLE, simple->
      cred, ldap->sctrlsp, NULL, NULL);
00321         }
00322
00323     objunref(ldap);
00324     return res;
00325 }
00326
00335 extern struct ldap_conn *ldap_connect(const char *uri, enum
      ldap_starttls starttls, int timelimit, int limit, int debug, int *err) {
00336     struct ldap_conn *ld;
00337     int version = 3;
00338     int res, sslres;
```

```
00339     struct timeval timeout;
00340
00341     if (!(ld = objalloc(sizeof(*ld), free_ldapconn))) {
00342         return NULL;
00343     }
00344
00345     ld->uri = strdup(uri);
00346     ld->sctrlsp = NULL;
00347     ld->timelim = timelimit;
00348     ld->limit = limit;
00349     ld->sasl = NULL;
00350
00351     if ((res = ldap_initialize(&ld->ldap, ld->uri) != LDAP_SUCCESS)) {
00352         objunref(ld);
00353         ld = NULL;
00354     } else {
00355         if (debug) {
00356             ldap_set_option(NULL, LDAP_OPT_DEBUG_LEVEL, &debug);
00357             ber_set_option(NULL, LBER_OPT_DEBUG_LEVEL, &debug);
00358         }
00359         if (timelimit) {
00360             timeout.tv_sec = timelimit;
00361             timeout.tv_usec = 0;
00362             ldap_set_option(ld->ldap, LDAP_OPT_NETWORK_TIMEOUT, (void *)&timeout);
00363         }
00364         ldap_set_option(ld->ldap, LDAP_OPT_PROTOCOL_VERSION, &version);
00365         ldap_set_option(ld->ldap, LDAP_OPT_REFERRALS, (void *)LDAP_OPT_ON);
00366         ldap_set_rebind_proc(ld->ldap, ldap_rebind_proc, ld);
00367
00368         if ((starttls != LDAP_STARTTLS_NONE) & !ldap_tls_inplace(ld->
      ldap) && (sslres = ldap_start_tls_s(ld->ldap, ld->sctrlsp, NULL))) {
00369             if (starttls == LDAP_STARTTLS_ENFORCE) {
00370                 objunref(ld);
00371                 ld = NULL;
00372                 res = sslres;
00373             }
00374         }
00375     }
00376     *err = res;
00377     return ld;
00378 }
00379
00380 static int interaction(unsigned flags, sasl_interact_t *interact, struct
      sasl_defaults *defaults) {
00381     const char *res = interact->defresult;
00382
00383     switch( interact->id ) {
00384         case SASL_CB_GETREALM:
00385             if (defaults->realm) {
00386                 res = defaults->realm;
00387             }
00388             break;
00389         case SASL_CB_AUTHNAME:
00390             if (defaults->authcid) {
00391                 res = defaults->authcid;
00392             }
00393             break;
00394         case SASL_CB_PASS:
00395             if (defaults->passwd) {
00396                 res = defaults->passwd;
00397             }
00398             break;
00399         case SASL_CB_USER:
00400             if (defaults->authzid) {
00401                 res = defaults->authzid;
00402             }
00403             break;
00404     }
00405
00406     interact->result = (res) ? res : "";
00407     interact->len = strlen(interact->result);
00408
00409     return LDAP_SUCCESS;
00410 }
00411
00412 static int dts_sasl_interact(LDAP *ld, unsigned flags, void *defaults, void *in ) {
00413     sasl_interact_t *interact = in;
00414
00415     if (!ld) {
00416         return LDAP_PARAM_ERROR;
00417     }
00418
00419     while( interact->id != SASL_CB_LIST_END ) {
00420         int rc = interaction(flags, interact, defaults);
00421         if (rc)  {
00422             return rc;
00423         }
```

```
00424         interact++;
00425     }
00426     return LDAP_SUCCESS;
00427 }
00428
00434 extern int ldap_simplebind(struct ldap_conn *ld, const char *dn, const char *passwd
      ) {
00435     struct ldap_simple *simple;
00436     struct berval *cred;
00437     int res, len = 0;
00438
00439     if (!objref(ld)) {
00440         return LDAP_UNAVAILABLE;
00441     }
00442
00443     if (passwd) {
00444         len = strlen(passwd);
00445     }
00446     simple = objalloc(sizeof(*simple), free_simple);
00447     cred = calloc(sizeof(*cred), 1);
00448     cred->bv_val = malloc(len);
00449     memcpy(cred->bv_val, passwd, len);
00450     cred->bv_len=len;
00451     simple->cred = cred;
00452     simple->dn = strdup(dn);
00453
00454     objlock(ld);
00455     if (ld->simple) {
00456         objunref(ld->simple);
00457     }
00458     ld->simple = simple;
00459     res = ldap_sasl_bind_s(ld->ldap, simple->dn, LDAP_SASL_SIMPLE, simple->
      cred, ld->sctrlsp, NULL, NULL);
00460     objunlock(ld);
00461     objunref(ld);
00462     return res;
00463 }
00464
00478 extern int ldap_simplerebind(struct ldap_conn *ldap, const char *initialdn, const
       char *initialpw, const char *base, const char *filter,
00479                              const char *uidrdn, const char *uid, const char *passwd) {
00480     int res, flen;
00481     struct ldap_results *results;
00482     const char *sfilt;
00483
00484     if (!objref(ldap)) {
00485         return LDAP_UNAVAILABLE;
00486     }
00487
00488     if ((res = ldap_simplebind(ldap, initialdn, initialpw))) {
00489         objunref(ldap);
00490         return res;
00491     }
00492
00493     flen=strlen(uidrdn) + strlen(filter) + strlen(uid) + 7;
00494     sfilt = malloc(flen);
00495     snprintf((char *)sfilt, flen, "(&(%s=%s)%s)", uidrdn, uid, filter);
00496
00497     if (!(results = ldap_search_sub(ldap, base, sfilt, 0, &res, uidrdn, NULL))) {
00498         free((void *)sfilt);
00499         objunref(ldap);
00500         return res;
00501     }
00502     free((void *)sfilt);
00503
00504     if (results->count != 1) {
00505         objunref(results);
00506         objunref(ldap);
00507         return LDAP_INAPPROPRIATE_AUTH;
00508     }
00509
00510     res = ldap_simplebind(ldap, results->first_entry->
      dn, passwd);
00511     objunref(ldap);
00512     objunref(results);
00513     return res;
00514 }
00515
00524 extern int ldap_saslbind(struct ldap_conn *ld, const char *mech, const char *realm,
      const char *authcid, const char *passwd, const char *authzid ) {
00525     struct sasl_defaults *sasl;
00526     int res, sasl_flags = LDAP_SASL_AUTOMATIC | LDAP_SASL_QUIET;
00527
00528     if (!objref(ld)) {
00529         return LDAP_UNAVAILABLE;
00530     }
00531
```

```
00532     if (!(sasl = objalloc(sizeof(*sasl), free_sasl))) {
00533         return LDAP_NO_MEMORY;
00534     }
00535 
00536     ALLOC_CONST(sasl->passwd, passwd);
00537 
00538     if (mech) {
00539         ALLOC_CONST(sasl->mech, mech);
00540     } else {
00541         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_MECH, &sasl->mech);
00542     }
00543 
00544     if (realm) {
00545         ALLOC_CONST(sasl->realm, realm);
00546     } else {
00547         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_REALM, &sasl->realm );
00548     }
00549 
00550     if (authcid) {
00551         ALLOC_CONST(sasl->authcid, authcid);
00552     } else {
00553         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHCID, &sasl->authcid );
00554     }
00555 
00556     if (authzid) {
00557         ALLOC_CONST(sasl->authzid, authzid);
00558     } else {
00559         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHZID, &sasl->authzid );
00560     }
00561 
00562     objlock(ld);
00563     if (ld->sasl) {
00564         objunref(ld->sasl);
00565     }
00566     ld->sasl = sasl;
00567     res = ldap_sasl_interactive_bind_s(ld->ldap, NULL, sasl->mech, ld->
      sctrlsp , NULL, sasl_flags, dts_sasl_interact, sasl);
00568     objunlock(ld);
00569     objunref(ld);
00570     return res;
00571 }
00572 
00576 extern const char *ldap_errmsg(int res) {
00577     return ldap_err2string(res);
00578 }
00579 
00580 static int32_t searchresults_hash(const void *data, int key) {
00581     int ret;
00582     const struct ldap_entry *ent = data;
00583     const char *hashkey = (key) ? data : ent->dn;
00584 
00585     if (hashkey) {
00586         ret = jenhash(hashkey, strlen(hashkey), 0);
00587     } else {
00588         ret = jenhash(ent, sizeof(ent), 0);
00589     }
00590     return(ret);
00591 }
00592 
00601 extern struct ldap_results *ldap_search_sub(struct
      ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res, ...) {
00602     va_list a_list;
00603     char *attr, **tmp, **attrs = NULL;
00604     int cnt = 1;
00605 
00606     va_start(a_list, res);
00607     while (( attr=va_arg(a_list, void *))) {
00608         cnt++;
00609     }
00610     va_end(a_list);
00611 
00612     if (cnt > 1) {
00613         tmp = attrs = malloc(sizeof(void *)*cnt);
00614 
00615         va_start(a_list, res);
00616         while (( attr=va_arg(a_list, char *))) {
00617             *tmp = attr;
00618             tmp++;
00619         }
00620         va_end(a_list);
00621         *tmp=NULL;
00622     }
00623 
00624     return _dtsldapsearch(ld, base, LDAP_SCOPE_SUBTREE, filter, attrs, b64enc, res);
00625 }
00626 
00635 extern struct ldap_results *ldap_search_one(struct
```

```
      ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res, ...) {
00636     va_list a_list;
00637     char *attr, **tmp, **attrs = NULL;
00638     int cnt = 1;
00639 
00640     va_start(a_list, res);
00641     while (( attr=va_arg(a_list, void *))) {
00642         cnt++;
00643     }
00644     va_end(a_list);
00645 
00646     if (cnt > 1) {
00647         tmp = attrs = malloc(sizeof(void *)*cnt);
00648 
00649         va_start(a_list, res);
00650         while (( attr=va_arg(a_list, char *))) {
00651             *tmp = attr;
00652             tmp++;
00653         }
00654         va_end(a_list);
00655         *tmp=NULL;
00656     }
00657 
00658     return _dtsldapsearch(ld, base, LDAP_SCOPE_ONELEVEL, filter, attrs, b64enc, res);
00659 }
00660 
00669 extern struct ldap_results *ldap_search_base(struct
      ldap_conn *ld, const char *base, const char *filter, int b64enc, int *res, ...) {
00670     va_list a_list;
00671     char *attr, **tmp, **attrs = NULL;
00672     int cnt = 1;
00673 
00674     va_start(a_list, res);
00675     while (( attr=va_arg(a_list, void *))) {
00676         cnt++;
00677     }
00678     va_end(a_list);
00679 
00680     if (cnt > 1) {
00681         tmp = attrs = malloc(sizeof(void *)*cnt);
00682 
00683         va_start(a_list, res);
00684         while (( attr=va_arg(a_list, char *))) {
00685             *tmp = attr;
00686             tmp++;
00687         }
00688         va_end(a_list);
00689         *tmp=NULL;
00690     }
00691 
00692     return _dtsldapsearch(ld, base, LDAP_SCOPE_BASE, filter, attrs, b64enc, res);
00693 }
00694 
00695 
00696 int ldap_count(LDAP *ld, LDAPMessage *message, int *err) {
00697     int x;
00698 
00699     objlock(ld);
00700     x = ldap_count_entries(ld, message);
00701     objunlock(ld);
00702 
00703     if (!err) {
00704         return x;
00705     }
00706 
00707     if (x < 0) {
00708         objlock(ld);
00709         ldap_get_option(ld, LDAP_OPT_RESULT_CODE, err);
00710         objunlock(ld);
00711     } else {
00712         *err = LDAP_SUCCESS;
00713     }
00714     return x;
00715 }
00716 
00717 static char *ldap_getdn(LDAP *ld, LDAPMessage *message, int *err) {
00718     char *dn;
00719 
00720     objlock(ld);
00721     dn = ldap_get_dn(ld, message);
00722     objunlock(ld);
00723 
00724     if (!err) {
00725         return dn;
00726     }
00727 
00728     if (!dn) {
```

```
00729         objlock(ld);
00730         ldap_get_option(ld, LDAP_OPT_RESULT_CODE, err);
00731         objunlock(ld);
00732     } else {
00733         *err = LDAP_SUCCESS;
00734     }
00735
00736     return dn;
00737 }
00738
00739 static char *ldap_getattribute(LDAP *ld, LDAPMessage *message, BerElement **berptr, int *err) {
00740     BerElement *ber = *berptr;
00741     char *attr = NULL;
00742
00743     objlock(ld);
00744     if (ber) {
00745         attr = ldap_next_attribute(ld, message, ber);
00746     } else {
00747         attr = ldap_first_attribute(ld, message, berptr);
00748     }
00749     if (!err) {
00750         objunlock(ld);
00751         return attr;
00752     }
00753
00754     if (!attr) {
00755         ldap_get_option(ld, LDAP_OPT_RESULT_CODE, err);
00756     } else {
00757         *err = LDAP_SUCCESS;
00758     }
00759
00760     objunlock(ld);
00761     return attr;
00762 }
00763
00764 static char *ldap_encattr(void *attrval, int b64enc, enum ldap_attrtype *type) {
00765     struct berval *val = attrval;
00766     char *aval = NULL;
00767     int len, pos, atype;
00768
00769     len = val->bv_len;
00770     for(pos=0; isprint(val->bv_val[pos]); pos++)
00771         ;
00772     if (pos == len) {
00773         aval = objalloc(val->bv_len+1, NULL);
00774         strncpy(aval, val->bv_val, objsize(aval));
00775         atype = LDAP_ATTRTYPE_CHAR;
00776     } else
00777         if (b64enc) {
00778             aval = b64enc_buf(val->bv_val, val->bv_len, 0);
00779             atype = LDAP_ATTRTYPE_B64;
00780         } else {
00781             aval = objalloc(val->bv_len, NULL);
00782             memcpy(aval, val->bv_val, objsize(aval));
00783             atype = LDAP_ATTRTYPE_OCTET;
00784         }
00785
00786     if (type) {
00787         *type = atype;
00788     }
00789
00790     return aval;
00791 }
00792
00793 static struct berval **ldap_attrvals(LDAP *ld, LDAPMessage *message, char *attr, int *cnt, int *err) {
00794     struct berval **vals = NULL;
00795
00796     objlock(ld);
00797     vals = ldap_get_values_len(ld, message, attr);
00798     objunlock(ld);
00799
00800     if (cnt) {
00801         *cnt = ldap_count_values_len(vals);
00802     }
00803
00804     if (!err) {
00805         return vals;
00806     }
00807
00808     if (!vals) {
00809         ldap_get_option(ld, LDAP_OPT_RESULT_CODE, err);
00810     } else {
00811         *err = LDAP_SUCCESS;
00812     }
00813
00814     return vals;
00815 }
```

```
00816
00817 static int32_t ldapattr_hash(const void *data, int key) {
00818     int ret;
00819     const struct ldap_attr *la = data;
00820     const char *hashkey = (key) ? data : la->name;
00821
00822     if (hashkey) {
00823         ret = jenhash(hashkey, strlen(hashkey), 0);
00824     } else {
00825         ret = jenhash(la, sizeof(la), 0);
00826     }
00827     return(ret);
00828 }
00829
00830 static struct bucket_list *attr2bl(LDAP *ld, LDAPMessage *message, struct
      ldap_attr **first, int b64enc, int *res) {
00831     BerElement *ber = NULL;
00832     struct bucket_list *bl;
00833     struct ldap_attr *la, *prev = NULL;
00834     struct ldap_attrval *lav, **lavals;
00835     struct berval **tmp, **vals = NULL;
00836     enum ldap_attrtype type;
00837     char *attr;
00838     int cnt;
00839     char *eval;
00840
00841     if (!(bl = create_bucketlist(4, ldapattr_hash))) {
00842         if (res) {
00843             *res = LDAP_NO_MEMORY;
00844         }
00845         return NULL;
00846     }
00847
00848     while((attr = ldap_getattribute(ld, message, &ber, res))) {
00849         tmp = vals = ldap_attrvals(ld, message, attr, &cnt, res);
00850         la = objalloc(sizeof(*la), free_attr);
00851         if (first && !*first) {
00852             *first = la;
00853         }
00854         la->next = NULL;
00855         if (prev) {
00856             prev->next = la;
00857             la->prev = prev;
00858         } else {
00859             la->prev = NULL;
00860         }
00861         prev = la;
00862         lavals = objalloc(sizeof(void *) * (cnt+1), free_attrvalarr);
00863         if (!lavals || !la) {
00864             if (res) {
00865                 *res = LDAP_NO_MEMORY;
00866             }
00867             if (la) {
00868                 objunref(la);
00869             }
00870             if (lavals) {
00871                 objunref(lavals);
00872             }
00873             objunref(bl);
00874             ldap_value_free_len(vals);
00875             if (ber) {
00876                 ber_free(ber, 0);
00877             }
00878             return NULL;
00879         }
00880         la->vals = lavals;
00881         la->name = attr;
00882         la->count = cnt;
00883
00884         for(; *tmp; tmp++) {
00885             struct berval *bval = *tmp;
00886
00887             *lavals = lav = objalloc(sizeof(*lav), free_attrval);
00888             lavals++;
00889
00890             eval = ldap_encattr(bval, b64enc, &type);
00891             if (!eval || !lav) {
00892                 if (res) {
00893                     *res = LDAP_NO_MEMORY;
00894                 }
00895                 objunref(bl);
00896                 objunref(la);
00897                 if (eval) {
00898                     objunref(eval);
00899                 }
00900                 ldap_value_free_len(vals);
00901                 if (ber) {
```

```
00902                     ber_free(ber, 0);
00903                 }
00904                 return NULL;
00905             }
00906             lav->len = bval->bv_len;
00907             lav->buffer = eval;
00908             lav->type = type;
00909         }
00910         *lavals = NULL;
00911         ldap_value_free_len(vals);
00912         addtobucket(bl, la);
00913         objunref(la);
00914     }
00915     if (ber) {
00916         ber_free(ber, 0);
00917     }
00918     return bl;
00919 }
00920
00921 struct ldap_entry *ldap_getent(LDAP *ld, LDAPMessage **msgptr, LDAPMessage *result, int b64enc,
      int *err) {
00922     LDAPMessage *message = *msgptr;
00923     struct ldap_entry *ent = NULL;
00924     struct ldap_rdn *lrdn, *prev = NULL, *first = NULL;
00925     struct ldap_rdn **rdns;
00926     LDAPDN dnarr;
00927     LDAPRDN rdnarr;
00928     LDAPAVA *rdn;
00929     int res, cnt, tlen=0, dccnt=0;
00930
00931     objlock(ld);
00932     if (message) {
00933         message = ldap_next_entry(ld, message);
00934     } else {
00935         message = ldap_first_entry(ld, result);
00936     }
00937     *msgptr = message;
00938     objunlock(ld);
00939
00940     if (message && !(ent = objalloc(sizeof(*ent), free_entry))) {
00941         if (!err) {
00942             *err = LDAP_NO_MEMORY;
00943         }
00944         return NULL;
00945     } else
00946         if (!message) {
00947             if (err) {
00948                 objlock(ld);
00949                 ldap_get_option(ld, LDAP_OPT_RESULT_CODE, err);
00950                 objunlock(ld);
00951             }
00952             return NULL;
00953         }
00954
00955     if (!(ent->dn = ldap_getdn(ld, message, &res))) {
00956         if (err) {
00957             *err = res;
00958         }
00959         objunref(ent);
00960         return NULL;
00961     }
00962
00963     objlock(ld);
00964     if ((res = ldap_str2dn(ent->dn, &dnarr, LDAP_DN_PEDANTIC))) {
00965         objunlock(ld);
00966         if (err) {
00967             *err = res;
00968         }
00969         objunref(ent);
00970         return NULL;
00971     }
00972     objunlock(ld);
00973
00974     ent->rdncnt = 0;
00975     for (cnt=0; dnarr[cnt]; cnt++) {
00976         rdnarr = dnarr[cnt];
00977         for (; *rdnarr; rdnarr++) {
00978             if (!(lrdn = objalloc(sizeof(*lrdn), free_rdn))) {
00979                 for(lrdn = first; lrdn; lrdn=lrdn->next) {
00980                     objunref(lrdn);
00981                 }
00982                 objunref(ent);
00983                 if (err) {
00984                     *err = LDAP_NO_MEMORY;
00985                 }
00986                 return NULL;
00987             }
```

```
00988
00989             ent->rdncnt++;
00990
00991             if (!first) {
00992                 first = lrdn;
00993             }
00994
00995             rdn = *rdnarr;
00996             ALLOC_CONST(lrdn->name, rdn->la_attr.bv_val);
00997             ALLOC_CONST(lrdn->value, rdn->la_value.bv_val);
00998
00999             if (!strcmp("dc", rdn->la_attr.bv_val)) {
01000                 dccnt++;
01001             }
01002             tlen += rdn->la_value.bv_len;
01003             lrdn->next = NULL;
01004             if (prev) {
01005                 prev->next = lrdn;
01006                 lrdn->prev = prev;
01007             } else {
01008                 lrdn->prev = NULL;
01009             }
01010             prev = lrdn;
01011         }
01012     }
01013     ldap_dnfree(dnarr);
01014
01015     ent->dnufn = calloc(tlen + (ent->rdncnt-dccnt)*2+dccnt, 1);
01016     ent->rdn = rdns = objalloc(sizeof(void *) * (ent->rdncnt+1), free_rdnarr);
01017
01018     if (!ent->dnufn || !ent->rdn) {
01019         for(lrdn = first; lrdn; lrdn=lrdn->next) {
01020             objunref(lrdn);
01021         }
01022         objunref(ent);
01023         if (err) {
01024             *err = LDAP_NO_MEMORY;
01025         }
01026     }
01027
01028     for(lrdn = first; lrdn ; lrdn = lrdn->next) {
01029         strcat((char *)ent->dnufn, lrdn->value);
01030         if (lrdn->next && !strcmp(lrdn->name, "dc")) {
01031             strcat((char *)ent->dnufn, ".");
01032         } else
01033             if (lrdn->next) {
01034                 strcat((char *)ent->dnufn, ", ");
01035             }
01036         *rdns = lrdn;
01037         rdns++;
01038     }
01039     *rdns = NULL;
01040
01041     if (!(ent->attrs = attr2bl(ld, message, &ent->first_attr, b64enc, &res))) {
01042         if (err) {
01043             *err = res;
01044         }
01045         objunref(ent);
01046         return NULL;
01047     }
01048
01049     if (err) {
01050         *err = LDAP_SUCCESS;
01051     }
01052
01053     return ent;
01054 }
01055
01059 extern void ldap_unref_attr(struct ldap_entry *entry, struct
      ldap_attr *attr) {
01060     if (!entry || !attr) {
01061         return;
01062     }
01063
01064     if (objcnt(attr) > 1) {
01065         objunref(attr);
01066     } else {
01067         if (attr == entry->first_attr) {
01068             entry->first_attr = attr->next;
01069         }
01070         remove_bucket_item(entry->attrs, attr);
01071     }
01072 }
01073
01077 extern void ldap_unref_entry(struct ldap_results *results, struct
      ldap_entry *entry) {
01078     if (!results || !entry) {
```

```
01079         return;
01080     }
01081
01082     if (objcnt(entry) > 1) {
01083         objunref(entry);
01084     } else {
01085         if (entry == results->first_entry) {
01086             results->first_entry = entry->next;
01087         }
01088         remove_bucket_item(results->entries, entry);
01089     }
01090 }
01091
01096 extern struct ldap_entry *ldap_getentry(struct
      ldap_results *results, const char *dn) {
01097     if (!results || !dn) {
01098         return NULL;
01099     }
01100     return (struct ldap_entry *)bucket_list_find_key(results->
      entries, dn);
01101 }
01102
01103
01108 extern struct ldap_attr *ldap_getattr(struct ldap_entry *entry, const char *
      attr) {
01109     if (!entry || !entry->attrs) {
01110         return NULL;
01111     }
01112     return (struct ldap_attr *)bucket_list_find_key(entry->
      attrs, attr);
01113 }
01114
01118 extern struct ldap_modify *ldap_modifyinit(const char *dn) {
01119     struct ldap_modify *mod;
01120     int cnt;
01121
01122     if (!(mod = objalloc(sizeof(*mod), free_modify))) {
01123         return NULL;
01124     }
01125
01126     ALLOC_CONST(mod->dn, dn);
01127     if (!mod->dn) {
01128         objunref(mod);
01129         return NULL;
01130     }
01131
01132     for(cnt=0; cnt < 3; cnt++) {
01133         if (!(mod->bl[cnt] = create_bucketlist(4, modify_hash))) {
01134             objunref(mod);
01135             return NULL;
01136         }
01137     }
01138
01139     return mod;
01140 }
01141
01142 static struct ldap_modreq *new_modreq(struct bucket_list *modtype, const char *attr)
      {
01143     struct ldap_modreq *modr;
01144
01145     if (!(modr = objalloc(sizeof(*modr), free_modreq))) {
01146         return NULL;
01147     }
01148
01149     ALLOC_CONST(modr->attr, attr);
01150     if (!modr->attr || !addtobucket(modtype, modr)) {
01151         objunref(modr);
01152         modr = NULL;
01153     }
01154     return modr;
01155 }
01156
01157 static struct ldap_modreq *getmodreq(struct ldap_modify *lmod, const char *attr, int
      modop) {
01158     struct bucket_list *bl = NULL;
01159     struct ldap_modreq *modr = NULL;
01160
01161     switch (modop) {
01162         case LDAP_MOD_REPLACE:
01163             bl = lmod->bl[0];
01164             break;
01165         case LDAP_MOD_DELETE:
01166             bl = lmod->bl[1];
01167             break;
01168         case LDAP_MOD_ADD:
01169             bl = lmod->bl[2];
01170             break;
```

```
01171     }
01172
01173     if (bl && !(modr = bucket_list_find_key(bl, attr))) {
01174         if (!(modr = new_modreq(bl, attr))) {
01175             return NULL;
01176         }
01177     }
01178     return modr;
01179 }
01180
01181 static int add_modifyval(struct ldap_modreq *modr, const char *value) {
01182     struct ldap_modval *newval;
01183
01184     if (!(newval = objalloc(sizeof(*newval), free_modval))) {
01185         return 1;
01186     }
01187
01188     ALLOC_CONST(newval->value, value);
01189     if (!newval->value) {
01190         objunref(newval);
01191         return 1;
01192     }
01193
01194     if (!modr->first) {
01195         modr->first = newval;
01196     }
01197     if (modr->last) {
01198         modr->last->next = newval;
01199     }
01200     modr->cnt++;
01201     modr->last = newval;
01202
01203     return 0;
01204 }
01205
01211 extern int ldap_mod_del(struct ldap_modify *lmod, const char *attr, ...) {
01212     va_list a_list;
01213     char *val;
01214     struct ldap_modreq *modr;
01215
01216     if (!(modr = getmodreq(lmod, attr, LDAP_MOD_DELETE))) {
01217         return 1;
01218     }
01219
01220     va_start(a_list, attr);
01221     while((val = va_arg(a_list, void *))) {
01222         if (add_modifyval(modr, val)) {
01223             objunref(modr);
01224             return(1);
01225         }
01226     }
01227
01228     objunref(modr);
01229     va_end(a_list);
01230     return 0;
01231 }
01232
01238 extern int ldap_mod_add(struct ldap_modify *lmod, const char *attr, ...) {
01239     va_list a_list;
01240     char *val;
01241     struct ldap_modreq *modr;
01242
01243     if (!(modr = getmodreq(lmod, attr, LDAP_MOD_ADD))) {
01244         return 1;
01245     }
01246
01247     va_start(a_list, attr);
01248     while((val = va_arg(a_list, void *))) {
01249         if (add_modifyval(modr, val)) {
01250             objunref(modr);
01251             return(1);
01252         }
01253     }
01254
01255     objunref(modr);
01256     va_end(a_list);
01257     return 0;
01258 }
01259
01265 extern int ldap_mod_rep(struct ldap_modify *lmod, const char *attr, ...) {
01266     va_list a_list;
01267     char *val;
01268     struct ldap_modreq *modr;
01269
01270     if (!(modr = getmodreq(lmod, attr, LDAP_MOD_REPLACE))) {
01271         return 1;
01272     }
```

```
01273
01274     va_start(a_list, attr);
01275     while((val = va_arg(a_list, void *))) {
01276         if (add_modifyval(modr, val)) {
01277             objunref(modr);
01278             return(1);
01279         }
01280     }
01281
01282     objunref(modr);
01283     va_end(a_list);
01284     return 0;
01285 }
01286
01287 static LDAPMod *ldap_reqtoarr(struct ldap_modreq *modr, int type) {
01288     LDAPMod *modi;
01289     const char **mval;
01290     struct ldap_modval *modv;
01291
01292     if (!(modi = calloc(sizeof(LDAPMod), 1))) {
01293         return NULL;
01294     }
01295
01296     if (!(modi->mod_values = calloc(sizeof(void *), modr->cnt+1))) {
01297         free(modi);
01298         return NULL;
01299     }
01300
01301     switch (type) {
01302         case 0:
01303             modi->mod_op = LDAP_MOD_REPLACE;
01304             break;
01305         case 1:
01306             modi->mod_op = LDAP_MOD_DELETE;
01307             break;
01308         case 2:
01309             modi->mod_op = LDAP_MOD_ADD;
01310             break;
01311         default
01312                 :
01313             modi->mod_op = 0;
01314             break;
01315     }
01316
01317     if (!(modi->mod_type = strdup(modr->attr))) {
01318         free(modi);
01319         return NULL;
01320     }
01321
01322     mval = (const char **)modi->mod_values;
01323     for(modv = modr->first; modv; modv=modv->next) {
01324         if (!(*mval = strdup(modv->value))) {
01325             ldap_mods_free(&modi, 0);
01326             return NULL;
01327         }
01328         mval++;
01329     }
01330     *mval = NULL;
01331
01332     return modi;
01333 }
01334
01339 extern int ldap_domodify(struct ldap_conn *ld, struct
      ldap_modify *lmod) {
01340     struct bucket_loop *bloop;
01341     struct ldap_modreq *modr;
01342     LDAPMod **modarr, **tmp, *item;
01343     int cnt, tot=0, res;
01344
01345     if (!objref(ld)) {
01346         return LDAP_UNAVAILABLE;
01347     }
01348
01349     for(cnt = 0; cnt < 3; cnt++) {
01350         tot += bucket_list_cnt(lmod->bl[cnt]);
01351     }
01352     tmp = modarr = calloc(sizeof(void *), (tot+1));
01353
01354     for(cnt = 0; cnt < 3; cnt++) {
01355         bloop = init_bucket_loop(lmod->bl[cnt]);
01356         while(bloop && ((modr = next_bucket_loop(bloop)))) {
01357             if (!(item = ldap_reqtoarr(modr, cnt))) {
01358                 ldap_mods_free(modarr, 1);
01359                 objunref(ld);
01360                 return LDAP_NO_MEMORY;
01361             }
01362             *tmp = item;
```

```
01363             tmp++;
01364             objunref(modr);
01365         }
01366         objunref(bloop);
01367     }
01368     *tmp = NULL;
01369
01370     objlock(ld);
01371     res = ldap_modify_ext_s(ld->ldap, lmod->dn, modarr, ld->sctrlsp, NULL);
01372     objunlock(ld);
01373     ldap_mods_free(modarr, 1);
01374     objunref(ld);
01375     return res;
01376 }
01377
01384 extern int ldap_mod_delattr(struct ldap_conn *ldap, const char *dn, const char *
      attr, const char *value) {
01385     struct ldap_modify *lmod;
01386     int res;
01387
01388     if (!(lmod = ldap_modifyinit(dn))) {
01389         return LDAP_NO_MEMORY;
01390     }
01391     if (ldap_mod_del(lmod, attr, value, NULL)) {
01392         objunref(lmod);
01393         return LDAP_NO_MEMORY;
01394     }
01395
01396     res = ldap_domodify(ldap, lmod);
01397     objunref(lmod);
01398     return res;
01399 }
01400
01406 extern int ldap_mod_rematTr(struct ldap_conn *ldap, const char *dn, const char *
      attr) {
01407     return ldap_mod_delattr(ldap, dn, attr, NULL);
01408 }
01409
01416 extern int ldap_mod_addattr(struct ldap_conn *ldap, const char *dn, const char *
      attr, const char *value) {
01417     int res = 0;
01418     struct ldap_modify *lmod;
01419
01420     if (!(lmod = ldap_modifyinit(dn))) {
01421         return LDAP_NO_MEMORY;
01422     }
01423
01424     if (ldap_mod_add(lmod, attr, value, NULL)) {
01425         objunref(lmod);
01426         return LDAP_NO_MEMORY;
01427     }
01428
01429     res = ldap_domodify(ldap, lmod);
01430     objunref(lmod);
01431     return res;
01432 }
01433
01434
01441 extern int ldap_mod_repattr(struct ldap_conn *ldap, const char *dn, const char *
      attr, const char *value) {
01442     struct ldap_modify *lmod;
01443     int res;
01444
01445     if (!(lmod = ldap_modifyinit(dn))) {
01446         return LDAP_NO_MEMORY;
01447     }
01448
01449     if (ldap_mod_rep(lmod, attr, value, NULL)) {
01450         objunref(lmod);
01451         return LDAP_NO_MEMORY;
01452     }
01453
01454     res = ldap_domodify(ldap, lmod);
01455     objunref(lmod);
01456     return res;
01457 }
01458
01462 extern struct ldap_add *ldap_addinit(const char *dn) {
01463     struct ldap_add *mod;
01464
01465     if (!(mod = objalloc(sizeof(*mod), free_add))) {
01466         return NULL;
01467     }
01468
01469     ALLOC_CONST(mod->dn, dn);
01470     if (!mod->dn) {
01471         objunref(mod);
```

```
01472         return NULL;
01473     }
01474
01475     if (!(mod->bl = create_bucketlist(4, modify_hash))) {
01476         objunref(mod);
01477         return NULL;
01478     }
01479
01480     return mod;
01481 }
01482
01483 static struct ldap_modreq *getaddreq(struct ldap_add *ladd, const char *attr) {
01484     struct bucket_list *bl = ladd->bl;
01485     struct ldap_modreq *modr = NULL;
01486
01487     if (bl && !(modr = bucket_list_find_key(bl, attr))) {
01488         if (!(modr = new_modreq(bl, attr))) {
01489             return NULL;
01490         }
01491     }
01492     return modr;
01493 }
01494
01500 extern int ldap_add_attr(struct ldap_add *ladd, const char *attr, ...) {
01501     va_list a_list;
01502     char *val;
01503     struct ldap_modreq *modr;
01504
01505     if (!(modr = getaddreq(ladd, attr))) {
01506         return 1;
01507     }
01508
01509     va_start(a_list, attr);
01510     while((val = va_arg(a_list, void *))) {
01511         if (add_modifyval(modr, val)) {
01512             objunref(modr);
01513             return(1);
01514         }
01515     }
01516
01517     objunref(modr);
01518     va_end(a_list);
01519     return 0;
01520 }
01521
01526 extern int ldap_doadd(struct ldap_conn *ld, struct ldap_add *ladd) {
01527     struct bucket_loop *bloop;
01528     struct ldap_modreq *modr;
01529     LDAPMod **modarr, **tmp, *item;
01530     int tot=0, res;
01531
01532     tot = bucket_list_cnt(ladd->bl);
01533     tmp = modarr = calloc(sizeof(void *), (tot+1));
01534
01535     bloop = init_bucket_loop(ladd->bl);
01536     while(bloop && ((modr = next_bucket_loop(bloop)))) {
01537         if (!(item = ldap_reqtoarr(modr, -1))) {
01538             ldap_mods_free(modarr, 1);
01539             return LDAP_NO_MEMORY;
01540         }
01541         *tmp = item;
01542         tmp++;
01543         objunref(modr);
01544     }
01545     objunref(bloop);
01546     *tmp = NULL;
01547
01548     objlock(ld);
01549     res = ldap_modify_ext_s(ld->ldap, ladd->dn, modarr, ld->sctrlsp, NULL);
01550     objunlock(ld);
01551     ldap_mods_free(modarr, 1);
01552
01553     return res;
01554 }
01555
01556
01559 struct ldap_results *_dtsldapsearch(struct ldap_conn *ldap, const char *base, int
      scope, const char *filter, char **attrs, int b64enc, int *err) {
01560     struct timeval timeout;
01561     struct ldap_results *results;
01562     struct ldap_entry *lent, *prev = NULL;
01563     LDAPMessage *result, *message = NULL;
01564     int res = LDAP_SUCCESS;
01565
01566     if (!objref(ldap)) {
01567         if (err) {
01568             *err = LDAP_UNAVAILABLE;
```

```
01569         }
01570         if (attrs) {
01571             free(attrs);
01572         }
01573         return NULL;
01574     }
01575 
01576     if ((results = objalloc(sizeof(*results), free_result))) {
01577         results->entries = create_bucketlist(4, searchresults_hash);
01578     }
01579 
01580     timeout.tv_sec = ldap->timelim;
01581     timeout.tv_usec = 0;
01582 
01583     objlock(ldap);
01584     if (!results || !results->entries ||
01585         (res = ldap_search_ext_s(ldap->ldap, base, scope, filter, attrs, 0, ldap->
      sctrlsp, NULL, &timeout, ldap->limit, &result))) {
01586         objunlock(ldap);
01587         objunref(ldap);
01588         objunref(results);
01589         ldap_msgfree(result);
01590         if (err) {
01591             *err = (!results || !results->entries) ? LDAP_NO_MEMORY : res;
01592         }
01593         if (attrs) {
01594             free(attrs);
01595         }
01596         return NULL;
01597     }
01598     objunlock(ldap);
01599 
01600     if (attrs) {
01601         free(attrs);
01602     }
01603 
01604     if ((results->count = ldap_count(ldap->ldap, result, err)) < 0) {
01605         objunref(ldap);
01606         objunref(results);
01607         ldap_msgfree(result);
01608         return NULL;
01609     }
01610 
01611     while((lent = ldap_getent(ldap->ldap, &message, result, b64enc, err))) {
01612         if (!results->first_entry) {
01613             results->first_entry = lent;
01614         }
01615         if (!addtobucket(results->entries, lent)) {
01616             res = LDAP_NO_MEMORY;
01617             objunref(lent);
01618             break;
01619         }
01620         lent->next = NULL;
01621         if (prev) {
01622             prev->next = lent;
01623             lent->prev = prev;
01624         } else {
01625             lent->prev = NULL;
01626         }
01627         prev = lent;
01628         objunref(lent);
01629     }
01630     ldap_msgfree(result);
01631 
01632     if (err) {
01633         *err = res;
01634     }
01635 
01636     if (res) {
01637         objunref(results);
01638         results = NULL;
01639     }
01640 
01641     objunref(ldap);
01642     return results;
01643 }
```

14.40 src/radius.c File Reference

Simple radius client implementation.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
#include <uuid/uuid.h>
#include <openssl/md5.h>
#include "include/dtsapp.h"
```

Data Structures

- struct radius_packet

  Radius Packet.
- struct radius_session

  Radius session.
- struct radius_connection

  Radius connection.
- struct radius_server

  Radius Server.

Functions

- void addradattrint (struct radius_packet *packet, char type, unsigned int val)

  Add a integer attribute too the packet.
- void addradattrip (struct radius_packet *packet, char type, char *ipaddr)

  Add a integer attribute too the packet.
- void addradattrstr (struct radius_packet *packet, char type, char *str)

  Add a integer attribute too the packet.
- struct radius_packet * new_radpacket (unsigned char code)

  Create a new radius packet.
- void add_radserver (const char *ipaddr, const char *auth, const char *acct, const char *secret, int timeout)

  Add new radius server to list of servers.
- int send_radpacket (struct radius_packet *packet, const char *userpass, radius_cb read_cb, void *cb_data)

  Send radius packet.
- unsigned char * radius_attr_first (struct radius_packet *packet)

  Return first packet attribute.
- unsigned char * radius_attr_next (struct radius_packet *packet, unsigned char *attr)

  Return next packet attribute.

14.40.1 Detailed Description

Simple radius client implementation.

Definition in file radius.c.

14.41 radius.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00029 #include <string.h>
00030 #include <stdio.h>
00031 #include <stdlib.h>
00032 #include <unistd.h>
00033 #include <errno.h>
00034
00035 #ifdef __WIN32__
00036 #include <winsock2.h>
00037 #else
00038 #include <arpa/inet.h>
00039 #endif
00040 #include <uuid/uuid.h>
00041 #include <openssl/md5.h>
00042 #include "include/dtsapp.h"
00043
00045 struct radius_packet {
00048     unsigned char code;
00050     unsigned char id;
00052     unsigned short len;
00054     unsigned char token[RAD_AUTH_TOKEN_LEN];
00056     unsigned char attrs[RAD_AUTH_PACKET_LEN -
      RAD_AUTH_HDR_LEN];
00057 };
00058
00063 struct radius_session {
00065     unsigned short id;
00067     unsigned char request[RAD_AUTH_TOKEN_LEN];
00069     void    *cb_data;
00071     radius_cb read_cb;
00073     unsigned int olen;
00075     struct radius_packet *packet;
00077     struct timeval sent;
00079     const char *passwd;
00081     char    retries;
00083     char    minserver;
00084 };
00085
00089 struct radius_connection {
00091     struct fwsocket *socket;
00093     unsigned char id;
00095     struct radius_server *server;
00097     struct bucket_list *sessions;
00098 };
00099
00105 struct radius_server {
00107     const char  *name;
00109     const char  *authport;
00111     const char  *acctport;
00113     const char  *secret;
00115     unsigned char   id;
00117     int     timeout;
00119     struct timeval  service;
00121     struct bucket_list *connex;
00122 };
00123
00124 static struct bucket_list *servers = NULL;
00125
00126 static struct radius_connection *radconnect(struct
      radius_server *server);
00127
00128 static unsigned char *addradattr(struct radius_packet *packet, char type, unsigned char *val,
      char len) {
00129     unsigned char *data = packet->attrs + packet->len - RAD_AUTH_HDR_LEN;
00130
00131     if (!len) {
00132         return NULL;
```

```
00133     }
00134
00135     data[0] = type;
00136     data[1] = len + 2;
00137     if (val) {
00138         memcpy(data + 2, val, len);
00139     }
00140
00141     packet->len += data[1];
00142     return (data);
00143 }
00144
00149 extern void addradattrint(struct radius_packet *packet, char type, unsigned int
      val) {
00150     unsigned int tval;
00151
00152     tval = htonl(val);
00153     addradattr(packet, type, (unsigned char *)&tval, sizeof(tval));
00154 }
00155
00160 extern void addradattrip(struct radius_packet *packet, char type, char *ipaddr) {
00161     unsigned int tval;
00162
00163     tval = inet_addr(ipaddr);
00164     addradattr(packet, type, (unsigned char *)&tval, sizeof(tval));
00165 }
00166
00171 extern void addradattrstr(struct radius_packet *packet, char type, char *str) {
00172     addradattr(packet, type, (unsigned char *)str, strlen(str));
00173 }
00174
00175 static void addradattrpasswd(struct radius_packet *packet, const char *pw, const char *secret)
       {
00176     unsigned char pwbuff[RAD_MAX_PASS_LEN];
00177     unsigned char digest[RAD_AUTH_TOKEN_LEN];
00178     MD5_CTX c, old;
00179     int len, n, i;
00180
00181     len = strlen(pw);
00182     if (len > RAD_MAX_PASS_LEN) {
00183         len = RAD_MAX_PASS_LEN;
00184     }
00185
00186     memcpy(pwbuff, pw, len);
00187     memset(pwbuff+len, 0, RAD_MAX_PASS_LEN -len);
00188
00189     /* pad len to RAD_AUTH_TOKEN_LEN*/
00190     if (!len) {
00191         len = RAD_AUTH_TOKEN_LEN;
00192     }  else
00193         if (!(len & 0xf)) {
00194             len += 0xf;
00195             len &= ~0xf;
00196         }
00197
00198     MD5_Init(&c);
00199     MD5_Update(&c, secret, strlen(secret));
00200     old = c;
00201
00202     MD5_Update(&c, packet->token, RAD_AUTH_TOKEN_LEN);
00203     for (n = 0; n < len; n += RAD_AUTH_TOKEN_LEN) {
00204         if (n > 0) {
00205             c = old;
00206             MD5_Update(&c, pwbuff + n - RAD_AUTH_TOKEN_LEN,
      RAD_AUTH_TOKEN_LEN);
00207         }
00208         MD5_Final(digest, &c);
00209         for (i = 0; i < RAD_AUTH_TOKEN_LEN; i++) {
00210             pwbuff[i + n] ^= digest[i];
00211         }
00212     }
00213     addradattr(packet, RAD_ATTR_USER_PASSWORD, pwbuff, len);
00214 }
00215
00221 extern struct radius_packet *new_radpacket(unsigned char
      code) {
00222     struct radius_packet *packet;
00223
00224     if ((packet = malloc(sizeof(*packet)))) {
00225         memset(packet, 0, sizeof(*packet));
00226         packet->len = RAD_AUTH_HDR_LEN;
00227         packet->code = code;
00228         genrand(&packet->token, RAD_AUTH_TOKEN_LEN);
00229     }
00230     return (packet);
00231 }
00232
```

```
00233 static int32_t hash_session(const void *data, int key) {
00234     unsigned int ret;
00235     const struct radius_session *session = data;
00236     const unsigned char *hashkey = (key) ? data : &session->id;
00237
00238     ret = *hashkey << 24;
00239
00240     return (ret);
00241 }
00242
00243 static int32_t hash_connex(const void *data, int key) {
00244     int ret;
00245     const struct radius_connection *connex = data;
00246     const int *hashkey = (key) ? data : &connex->socket;
00247
00248     ret = *hashkey;
00249
00250     return (ret);
00251 }
00252
00253 static int32_t hash_server(const void *data, int key) {
00254     int ret;
00255     const struct radius_server *server = data;
00256     const unsigned char *hashkey = (key) ? data : &server->id;
00257
00258     ret = *hashkey;
00259
00260     return(ret);
00261 }
00262
00263 static void del_radserver(void *data) {
00264     struct radius_server *server = data;
00265
00266     if (server->name) {
00267         free((char *)server->name);
00268     }
00269     if (server->authport) {
00270         free((char *)server->authport);
00271     }
00272     if (server->acctport) {
00273         free((char *)server->acctport);
00274     }
00275     if (server->secret) {
00276         free((char *)server->secret);
00277     }
00278     if (server->connex) {
00279         objunref(server->connex);
00280     }
00281 }
00282
00289 extern void add_radserver(const char *ipaddr, const char *auth, const char *acct, const char *
    secret, int timeout) {
00290     struct radius_server *server;
00291
00292     if ((server = objalloc(sizeof(*server), del_radserver))) {
00293         ALLOC_CONST(server->name, ipaddr);
00294         ALLOC_CONST(server->authport, auth);
00295         ALLOC_CONST(server->acctport, acct);
00296         ALLOC_CONST(server->secret, secret);
00297         if (!servers) {
00298             servers = create_bucketlist(0, hash_server);
00299         }
00300         server->id = bucket_list_cnt(servers);
00301         server->timeout = timeout;
00302         gettimeofday(&server->service, NULL);
00303         addtobucket(servers, server);
00304     }
00305
00306     objunref(server);
00307 }
00308
00309 static void del_radsession(void *data) {
00310     struct radius_session *session = data;
00311
00312     if (session->passwd) {
00313         free((void *)session->passwd);
00314     }
00315     if (session->packet) {
00316         free(session->packet);
00317     }
00318 }
00319
00320 static struct radius_session *rad_session(struct radius_packet *packet, struct
    radius_connection *connex,
00321         const char *passwd, radius_cb read_cb, void *
    cb_data) {
00322     struct radius_session *session = NULL;
```

```
00323
00324     if ((session = objalloc(sizeof(*session), del_radsession))) {
00325         if (!connex->sessions) {
00326             connex->sessions = create_bucketlist(4, hash_session);
00327         }
00328         memcpy(session->request, packet->token, RAD_AUTH_TOKEN_LEN);
00329         session->id = packet->id;
00330         session->packet = packet;
00331         session->read_cb = read_cb;
00332         session->cb_data = cb_data;
00333         session->olen = packet->len;
00334         session->retries = 2;
00335         ALLOC_CONST(session->passwd, passwd);
00336         addtobucket(connex->sessions, session);
00337     }
00338     return (session);
00339 }
00340
00341 static int _send_radpacket(struct radius_packet *packet, const char *userpass, struct
      radius_session *hint,
00342                            radius_cb read_cb, void *cb_data) {
00343     int scnt;
00344     unsigned char *vector;
00345     unsigned short len;
00346     struct radius_server *server;
00347     struct radius_session *session;
00348     struct radius_connection *connex;
00349     struct bucket_loop *sloop, *cloop;
00350     struct timeval  curtime;
00351
00352
00353     gettimeofday(&curtime, NULL);
00354     sloop = init_bucket_loop(servers);
00355     objref(hint);
00356     while (sloop && (server = next_bucket_loop(sloop))) {
00357         objlock(server);
00358         if ((hint && (server->id <= hint->minserver)) ||
00359                 (server->service.tv_sec > curtime.tv_sec)) {
00360             objunlock(server);
00361             objunref(server);
00362             continue;
00363         }
00364         if (!server->connex) {
00365             connex = radconnect(server);
00366             objunref(connex);
00367             objunlock(server);
00368             objref(server);
00369         } else {
00370             objunlock(server);
00371         }
00372         cloop = init_bucket_loop(server->connex);
00373         while (cloop && (connex = next_bucket_loop(cloop))) {
00374             objlock(connex);
00375             if (connex->sessions && (bucket_list_cnt(connex->
      sessions) > 254)) {
00376                 objunlock(connex);
00377                 objunref(connex);
00378                 /* if im overflowing get next or add new*/
00379                 objlock(server);
00380                 if (!(connex = next_bucket_loop(cloop))) {
00381                     if ((connex = radconnect(server))) {
00382                         objunlock(server);
00383                         objref(server);
00384                     } else {
00385                         break;
00386                     }
00387                 } else {
00388                     objunlock(server);
00389                 }
00390                 objlock(connex);
00391             }
00392
00393             connex->id++;
00394             if (hint) {
00395                 packet = hint->packet;
00396                 session = hint;
00397                 packet->id = connex->id;
00398                 session->id = packet->id;
00399                 session->retries = 2;
00400                 if (!connex->sessions) {
00401                     connex->sessions = create_bucketlist(4, hash_session);
00402                 }
00403                 addtobucket(connex->sessions, session);
00404             } else {
00405                 packet->id = connex->id;
00406                 session = rad_session(packet, connex, userpass, read_cb, cb_data);
00407             }
```

```
00408             session->minserver = server->id;
00409             objunlock(connex);
00410 
00411             if (session->passwd) {
00412                 addradattrpasswd(packet, session->passwd,  server->secret);
00413             }
00414 
00415             vector = addradattr(packet, RAD_ATTR_MESSAGE, NULL,
      RAD_AUTH_TOKEN_LEN);
00416             len = packet->len;
00417             packet->len = htons(len);
00418             md5hmac(vector + 2, packet, len, server->secret, strlen(server->
      secret));
00419 
00420             scnt = send(connex->socket->sock, packet, len, 0);
00421             memset(packet->attrs + session->olen - RAD_AUTH_HDR_LEN, 0, len -
      session->olen);
00422             packet->len = session->olen;
00423 
00424             objunref(connex);
00425             if (len == scnt) {
00426                 session->sent = curtime;
00427                 objunref(session);
00428                 objunref(server);
00429                 objunref(hint);
00430                 objunref(cloop);
00431                 objunref(sloop);
00432                 return (0);
00433             } else {
00434                 remove_bucket_item(connex->sessions, session);
00435             }
00436         }
00437         objunref(server);
00438         objunref(cloop);
00439     }
00440     objunref(sloop);
00441     objunref(hint);
00442 
00443     return (-1);
00444 }
00445 
00452 extern int send_radpacket(struct radius_packet *packet, const char *userpass,
      radius_cb read_cb, void *cb_data) {
00453     return (_send_radpacket(packet, userpass, NULL, read_cb, cb_data));
00454 }
00455 
00456 static int resend_radpacket(struct radius_session *session) {
00457     return (_send_radpacket(NULL, NULL, session, NULL, NULL));
00458 }
00459 
00460 static void rad_resend(struct radius_connection *connex) {
00461     struct radius_session *session;
00462     struct bucket_loop *bloop;
00463     struct timeval tv;
00464     unsigned int tdiff, len, scnt;
00465     unsigned char *vector;
00466 
00467     gettimeofday(&tv, NULL);
00468 
00469     bloop=init_bucket_loop(connex->sessions);
00470     while (bloop && (session = next_bucket_loop(bloop))) {
00471         tdiff = tv.tv_sec - session->sent.tv_sec;
00472         if (tdiff > 3) {
00473             if (!session->retries) {
00474                 remove_bucket_loop(bloop);
00475                 resend_radpacket(session);
00476                 objunref(session);
00477                 continue;
00478             }
00479 
00480             if (session->passwd) {
00481                 addradattrpasswd(session->packet, session->passwd, connex->
      server->secret);
00482             }
00483 
00484             vector = addradattr(session->packet, RAD_ATTR_MESSAGE, NULL,
      RAD_AUTH_TOKEN_LEN);
00485             len = session->packet->len;
00486             session->packet->len = htons(len);
00487             md5hmac(vector + 2, session->packet, len, connex->
      server->secret, strlen(connex->server->secret));
00488 
00489             scnt = send(connex->socket->sock, session->packet, len, 0);
00490             memset(session->packet->attrs + session->olen -
      RAD_AUTH_HDR_LEN, 0, len - session->olen);
00491             session->packet->len = session->olen;
00492             session->sent = tv;
```

```
00493             session->retries--;
00494             if (scnt != len) {
00495                 remove_bucket_loop(bloop);
00496                 resend_radpacket(session);
00497                 objunref(session);
00498             }
00499         }
00500         objunref(session);
00501     }
00502     objunref(bloop);
00503 }
00504
00505 static void radius_recv(void **data) {
00506     struct radius_connection *connex = *data;
00507     struct radius_packet *packet;
00508     unsigned char buff[RAD_AUTH_PACKET_LEN];
00509     unsigned char rtok[RAD_AUTH_TOKEN_LEN];
00510     unsigned char rtok2[RAD_AUTH_TOKEN_LEN];
00511     struct radius_session *session;
00512     int chk, plen;
00513
00514     chk = recv(connex->socket->sock, buff, 4096, 0);
00515
00516     if (chk < 0) {
00517         if (errno == ECONNREFUSED) {
00518             printf("Connection Bad\n");
00519         }
00520     } else
00521         if (chk == 0) {
00522             objlock(connex->server);
00523             printf("Taking server off line for %is\n", connex->server->
    timeout);
00524             gettimeofday(&connex->server->service, NULL);
00525             connex->server->service.tv_sec += connex->server->
    timeout;
00526             objunlock(connex->server);
00527         }
00528
00529     packet = (struct radius_packet *)&buff;
00530     plen = ntohs(packet->len);
00531
00532     if ((chk < plen) || (chk <= RAD_AUTH_HDR_LEN)) {
00533         printf("OOps Did not get proper packet\n");
00534         return;
00535     }
00536
00537     memset(buff + plen, 0, RAD_AUTH_PACKET_LEN - plen);
00538
00539     if (!(session = bucket_list_find_key(connex->sessions, &packet->
    id))) {
00540         printf("Could not find session\n");
00541         return;
00542     }
00543
00544     memcpy(rtok, packet->token, RAD_AUTH_TOKEN_LEN);
00545     memcpy(packet->token, session->request, RAD_AUTH_TOKEN_LEN);
00546     md5sum2(rtok2, packet, plen, connex->server->secret, strlen(connex->
    server->secret));
00547
00548     if (md5cmp(rtok, rtok2)) {
00549         printf("Invalid Signature");
00550         return;
00551     }
00552
00553     if (session->read_cb) {
00554         packet->len = plen;
00555         session->read_cb(packet, session->cb_data);
00556     }
00557
00558     remove_bucket_item(connex->sessions, session);
00559     objunref(session);
00560 }
00561
00562 static void *rad_return(void *data) {
00563     struct radius_connection *connex = data;
00564     fd_set  rd_set, act_set;
00565     struct  timeval tv;
00566     int selfd;
00567
00568     FD_ZERO(&rd_set);
00569     FD_SET(connex->socket->sock, &rd_set);
00570
00571     while (framework_threadok()) {
00572         act_set = rd_set;
00573         tv.tv_sec = 0;
00574         tv.tv_usec = 200000;
00575
```

```
00576         selfd = select(connex->socket->sock + 1, &act_set, NULL, NULL, &tv);
00577
00578         if ((selfd < 0 && errno == EINTR) || (!selfd)) {
00579             rad_resend(connex);
00580             continue;
00581         } else
00582             if (selfd < 0) {
00583                 break;
00584             }
00585
00586         if (FD_ISSET(connex->socket->sock, &act_set)) {
00587             radius_recv(data);
00588         }
00589         rad_resend(connex);
00590     }
00591
00592     return NULL;
00593 }
00594
00595 static void del_radconnect(void *data) {
00596     struct radius_connection *connex = data;
00597
00598     objunref(connex->server);
00599     objunref(connex->sessions);
00600     objunref(connex->socket);
00601 }
00602
00603 static struct radius_connection *radconnect(struct
      radius_server *server) {
00604     struct radius_connection *connex;
00605     int val = 1;
00606
00607     if ((connex = objalloc(sizeof(*connex), del_radconnect))) {
00608         if ((connex->socket = udpconnect(server->name, server->
      authport, NULL))) {
00609             if (!server->connex) {
00610                 server->connex = create_bucketlist(0, hash_connex);
00611             }
00612             setsockopt(connex->socket->sock, SOL_IP, IP_RECVERR,(char *)&val, sizeof(val));
00613             connex->server = server;
00614             genrand(&connex->id, sizeof(connex->id));
00615             addtobucket(server->connex, connex);
00616             framework_mkthread(rad_return, NULL, NULL, connex, 0);
00617         }
00618     }
00619     return (connex);
00620 }
00621
00627 extern unsigned char *radius_attr_first(struct radius_packet *packet) {
00628     return (packet->attrs);
00629 }
00630
00635 extern unsigned char *radius_attr_next(struct radius_packet *packet, unsigned
      char *attr) {
00636     int offset = (packet->len - RAD_AUTH_HDR_LEN) - (attr - packet->
      attrs);
00637
00638     if (!(offset - attr[1])) {
00639         return NULL;
00640     }
00641
00642     return (attr + attr[1]);
00643 }
00644
```

14.42 src/refobj.c File Reference

Referenced Lockable Objects.

```
#include <pthread.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include "include/dtsapp.h"
```

Data Structures

- struct ref_obj

  Internal structure of all referenced objects.
- struct blist_obj

  Entry in a bucket list.
- struct bucket_list

  Bucket list, hold hashed objects in buckets.
- struct bucket_loop

  Bucket iterator.

Macros

- #define REFOBJ_MAGIC 0xdeadc0de

  Magic number stored as first field of all referenced objects.
- #define refobj_offset sizeof(struct ref_obj);

  The size of ref_obj is the offset for the data.

Functions

- void * objalloc (int size, objdestroy destructor)

  Allocate a referenced lockable object.
- int objref (void *data)

  Reference a object.
- int objunref (void *data)

  Drop reference held.
- int objcnt (void *data)

  Return current reference count.
- int objsize (void *data)

  Size requested for data.
- int objlock (void *data)

  Lock the reference.
- int objtrylock (void *data)

  Try lock a reference.
- int objunlock (void *data)

  Unlock a reference.
- void * objchar (const char *orig)

  Return a reference to copy of a buffer.
- void * create_bucketlist (int bitmask, blisthash hash_function)
- int addtobucket (struct bucket_list *blist, void *data)

  Add a reference to the bucketlist.
- void remove_bucket_item (struct bucket_list *blist, void *data)

  Remove and unreference a item from the list.
- int bucket_list_cnt (struct bucket_list *blist)

  Return number of items in the list.
- void * bucket_list_find_key (struct bucket_list *blist, const void *key)

  Find and return a reference to a item matching supplied key.
- void bucketlist_callback (struct bucket_list *blist, blist_cb callback, void *data2)

  Run a callback function on all items in the list.
- struct bucket_loop * init_bucket_loop (struct bucket_list *blist)

Create a bucket list iterator to safely iterate the list.

- void * next_bucket_loop (struct bucket_loop *bloop)

  Return a reference to the next item in the list this could be the first item.

- void remove_bucket_loop (struct bucket_loop *bloop)

  Safely remove a item from a list while iterating in a loop.

14.42.1 Detailed Description

Referenced Lockable Objects.

Definition in file refobj.c.

14.43 refobj.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003     http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00026 #include <pthread.h>
00027 #include <string.h>
00028 #include <stdlib.h>
00029 #include <stdint.h>
00030 #include "include/dtsapp.h"
00031
00032 /* add one for ref obj's*/
00034 #define REFOBJ_MAGIC        0xdeadc0de
00035
00036 /* ref counted objects*/
00038 struct ref_obj {
00042     uint32_t    magic;
00045     uint32_t    cnt;
00048     size_t      size;
00050     pthread_mutex_t lock;
00052     objdestroy    destroy;
00054     void        *data;
00055 };
00056
00061 struct blist_obj {
00064     int32_t     hash;
00066     struct      blist_obj *next;
00068     struct      blist_obj *prev;
00070     struct      ref_obj *data;
00071 };
00072
00075 struct bucket_list {
00077     unsigned short  bucketbits;
00079     size_t      count;
00081     blisthash  hash_func;
00083     struct      blist_obj **list;
00085     pthread_mutex_t *locks;
00087     size_t      *version;
00088 };
00089
00097 struct bucket_loop {
00099     struct bucket_list *blist;
00101     unsigned short bucket;
00104     size_t version;
00106     uint32_t head_hash;
00108     uint32_t cur_hash;
00110     struct blist_obj *head;
00112     struct blist_obj *cur;
00113 };
00114
```

```
00119 #define refobj_offset   sizeof(struct ref_obj);
00120
00129 extern void *objalloc(int size,objdestroy destructor) {
00130     struct ref_obj *ref;
00131     int asize;
00132     char *robj;
00133
00134     asize  = size + refobj_offset;
00135
00136     if ((robj = malloc(asize))) {
00137         memset(robj, 0, asize);
00138         ref = (struct ref_obj *)robj;
00139         pthread_mutex_init(&ref->lock, NULL);
00140         ref->magic = REFOBJ_MAGIC;
00141         ref->cnt = 1;
00142         ref->data = robj + refobj_offset;
00143         ref->size = asize;
00144         ref->destroy = destructor;
00145         return (ref->data);
00146     }
00147     return NULL;
00148 }
00149
00153 extern int objref(void *data) {
00154     char *ptr = data;
00155     struct ref_obj *ref;
00156     int ret = 0;
00157
00158     ptr = ptr - refobj_offset;
00159     ref = (struct ref_obj *)ptr;
00160
00161     if (!data || !ref || (ref->magic != REFOBJ_MAGIC)) {
00162         return (ret);
00163     }
00164
00165     /*double check just incase im gone*/
00166     if (!pthread_mutex_lock(&ref->lock)) {
00167         if ((ref->magic == REFOBJ_MAGIC) && (ref->cnt > 0)) {
00168             ref->cnt++;
00169             ret = ref->cnt;
00170         }
00171         pthread_mutex_unlock(&ref->lock);
00172     }
00173
00174     return (ret);
00175 }
00176
00184 extern int objunref(void *data) {
00185     char *ptr = data;
00186     struct ref_obj *ref;
00187     int ret = -1;
00188
00189     if (!data) {
00190         return (ret);
00191     }
00192
00193     ptr = ptr - refobj_offset;
00194     ref = (struct ref_obj *)ptr;
00195
00196     if ((ref->magic == REFOBJ_MAGIC) && (ref->cnt)) {
00197         pthread_mutex_lock(&ref->lock);
00198         ref->cnt--;
00199         ret = ref->cnt;
00200         /* free the object its no longer in use*/
00201         if (!ret) {
00202             ref->magic = 0;
00203             ref->size = 0;
00204             ref->data = NULL;
00205             if (ref->destroy) {
00206                 ref->destroy(data);
00207             }
00208             pthread_mutex_unlock(&ref->lock);
00209             pthread_mutex_destroy(&ref->lock);
00210             free(ref);
00211         } else {
00212             pthread_mutex_unlock(&ref->lock);
00213         }
00214     }
00215     return (ret);
00216 }
00217
00222 extern int objcnt(void *data) {
00223     char *ptr = data;
00224     int ret = -1;
00225     struct ref_obj *ref;
00226
00227     if (!data) {
```

```
00228         return (ret);
00229     }
00230
00231     ptr = ptr - refobj_offset;
00232     ref = (struct ref_obj *)ptr;
00233
00234     if (ref->magic == REFOBJ_MAGIC) {
00235         pthread_mutex_lock(&ref->lock);
00236         ret = ref->cnt;
00237         pthread_mutex_unlock(&ref->lock);
00238     }
00239     return (ret);
00240 }
00241
00246 extern int objsize(void *data) {
00247     char *ptr = data;
00248     int ret = 0;
00249     struct ref_obj *ref;
00250
00251     if (!data) {
00252         return (ret);
00253     }
00254
00255     ptr = ptr - refobj_offset;
00256     ref = (struct ref_obj *)ptr;
00257
00258     if (ref->magic == REFOBJ_MAGIC) {
00259         pthread_mutex_lock(&ref->lock);
00260         ret = ref->size - refobj_offset;
00261         pthread_mutex_unlock(&ref->lock);
00262     }
00263     return (ret);
00264 }
00265
00269 extern int objlock(void *data) {
00270     char *ptr = data;
00271     struct ref_obj *ref;
00272
00273     ptr = ptr - refobj_offset;
00274     ref = (struct ref_obj *)ptr;
00275
00276     if (data && ref->magic == REFOBJ_MAGIC) {
00277         pthread_mutex_lock(&ref->lock);
00278     }
00279     return (0);
00280 }
00281
00285 extern int objtrylock(void *data) {
00286     char *ptr = data;
00287     struct ref_obj *ref;
00288
00289     ptr = ptr - refobj_offset;
00290     ref = (struct ref_obj *)ptr;
00291
00292     if (ref->magic == REFOBJ_MAGIC) {
00293         return ((pthread_mutex_trylock(&ref->lock)) ? -1 : 0);
00294     }
00295     return (-1);
00296 }
00297
00301 extern int objunlock(void *data) {
00302     char *ptr = data;
00303     struct ref_obj *ref;
00304
00305     ptr = ptr - refobj_offset;
00306     ref = (struct ref_obj *)ptr;
00307
00308     if (ref->magic == REFOBJ_MAGIC) {
00309         pthread_mutex_unlock(&ref->lock);
00310     }
00311     return (0);
00312 }
00313
00314 static void empty_buckets(void *data) {
00315     struct bucket_list *blist = data;
00316     struct bucket_loop *bloop;
00317     void *entry;
00318
00319     bloop = init_bucket_loop(blist);
00320     while (bloop && (entry = next_bucket_loop(bloop))) {
00321         remove_bucket_loop(bloop);
00322         objunref(entry);
00323     }
00324     objunref(bloop);
00325 }
00326
00330 extern void *objchar(const char *orig) {
```

```
00331     int len = strlen(orig) + 1;
00332     void *nobj;
00333
00334     if ((nobj = objalloc(len, NULL))) {
00335         memcpy(nobj, orig, len);
00336     }
00337     return nobj;
00338 }
00339
00356 extern void *create_bucketlist(int bitmask, blisthash hash_function) {
00357     struct bucket_list *new;
00358     short int buckets, cnt;
00359
00360     buckets = (1 << bitmask);
00361
00362     /* allocate session bucket list memory size of the struct plus a list lock and version for each bucket
      */
00363     if (!(new = objalloc(sizeof(*new) + (sizeof(void *) + sizeof(pthread_mutex_t) + sizeof(size_t))
       * buckets, empty_buckets))) {
00364         return NULL;
00365     }
00366
00367     /*initialise each bucket*/
00368     new->bucketbits = bitmask;
00369     new->list = (void *)((char *)new + sizeof(*new));
00370     for (cnt = 0; cnt < buckets; cnt++) {
00371         if ((new->list[cnt] = malloc(sizeof(*new->list[cnt])))) {
00372             memset(new->list[cnt], 0, sizeof(*new->list[cnt]));
00373         }
00374     }
00375
00376     /*next pointer is pointer to locks*/
00377     new->locks = (void *)&new->list[buckets];
00378     for (cnt = 0; cnt < buckets; cnt++) {
00379         pthread_mutex_init(&new->locks[cnt], NULL);
00380     }
00381
00382     /*Next up version array*/
00383     new->version = (void *)&new->locks[buckets];
00384
00385     new->hash_func = hash_function;
00386
00387     return (new);
00388 }
00389
00390 static struct blist_obj *blist_gotohash(struct blist_obj *cur, unsigned int
      hash, int bucketbits) {
00391     struct blist_obj *lhead = cur;
00392
00393     if ((hash << bucketbits) < 0) {
00394         do {
00395             lhead = lhead->prev;
00396         } while ((lhead->hash > hash) && lhead->prev->next);
00397     } else {
00398         while (lhead && lhead->next && (lhead->next->hash < hash)) {
00399             lhead = lhead->next;
00400         }
00401     }
00402
00403     return (lhead);
00404 }
00405
00406 static int gethash(struct bucket_list *blist, const void *data, int key) {
00407     const char *ptr = data;
00408     struct ref_obj *ref;
00409     int hash = 0;
00410
00411     ptr = ptr - refobj_offset;
00412     ref = (struct ref_obj *)ptr;
00413
00414     if (blist->hash_func) {
00415         hash = blist->hash_func(data, key);
00416     } else if (ref && (ref->magic == REFOBJ_MAGIC)) {
00417         hash = jenhash(ref, ref->size, 0);
00418     }
00419     return (hash);
00420 }
00421
00428 extern int addtobucket(struct bucket_list *blist, void *data) {
00429     char *ptr = data;
00430     struct ref_obj *ref;
00431     struct blist_obj *lhead, *tmp;
00432     unsigned int hash, bucket;
00433
00434     if (!objref(blist)) {
00435         return (0);
00436     }
```

```
00437
00438     if (!objref(data)) {
00439         objunref(blist);
00440         return (0);
00441     }
00442
00443     ptr = ptr - refobj_offset;
00444     ref = (struct ref_obj *)ptr;
00445
00446     hash = gethash(blist, data, 0);
00447     bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
      bucketbits) - 1));
00448
00449     pthread_mutex_lock(&blist->locks[bucket]);
00450     lhead = blist->list[bucket];
00451     /*no head or non null head*/
00452     if (!lhead || lhead->prev) {
00453         if (!(tmp = malloc(sizeof(*tmp)))) {
00454             pthread_mutex_unlock(&blist->locks[bucket]);
00455             objunref(data);
00456             objunref(blist);
00457             return (0);
00458         }
00459         memset(tmp, 0, sizeof(*tmp));
00460         tmp->hash = hash;
00461         tmp->data = ref;
00462
00463         /*there is no head*/
00464         if (!lhead) {
00465             blist->list[bucket] = tmp;
00466             tmp->prev = tmp;
00467             tmp->next = NULL;
00468         /*become new head*/
00469         } else if (hash < lhead->hash) {
00470             tmp->next = lhead;
00471             tmp->prev = lhead->prev;
00472             lhead->prev = tmp;
00473             blist->list[bucket] = tmp;
00474         /*new tail*/
00475         } else if (hash > lhead->prev->hash) {
00476             tmp->prev = lhead->prev;
00477             tmp->next = NULL;
00478             lhead->prev->next = tmp;
00479             lhead->prev = tmp;
00480         /*insert entry*/
00481         } else {
00482             lhead = blist_gotohash(lhead, hash, blist->bucketbits);
00483             tmp->next = lhead->next;
00484             tmp->prev = lhead;
00485
00486             if (lhead->next) {
00487                 lhead->next->prev = tmp;
00488             } else {
00489                 blist->list[bucket]->prev = tmp;
00490             }
00491             lhead->next = tmp;
00492         }
00493     } else {
00494         /*set NULL head*/
00495         lhead->data = ref;
00496         lhead->prev = lhead;
00497         lhead->next = NULL;
00498         lhead->hash = hash;
00499     }
00500
00501     blist->version[bucket]++;
00502     pthread_mutex_unlock(&blist->locks[bucket]);
00503
00504     objlock(blist);
00505     blist->count++;
00506     objunlock(blist);
00507     objunref(blist);
00508
00509     return (1);
00510 }
00511
00517 extern void remove_bucket_item(struct bucket_list *blist, void *data) {
00518     struct blist_obj *entry;
00519     int hash, bucket;
00520
00521     hash = gethash(blist, data, 0);
00522     bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
      bucketbits) - 1));
00523
00524     pthread_mutex_lock(&blist->locks[bucket]);
00525     entry = blist_gotohash(blist->list[bucket], hash + 1, blist->bucketbits);
00526     if (entry && entry->hash == hash) {
```

```
00527         if (entry->next && (entry == blist->list[bucket])) {
00528             entry->next->prev = entry->prev;
00529             blist->list[bucket] = entry->next;
00530         } else if (entry->next) {
00531             entry->next->prev = entry->prev;
00532             entry->prev->next = entry->next;
00533         } else if (entry == blist->list[bucket]) {
00534             blist->list[bucket] = NULL;
00535         } else {
00536             entry->prev->next = NULL;
00537             blist->list[bucket]->prev = entry->prev;
00538         }
00539         objunref(entry->data->data);
00540         free(entry);
00541         objlock(blist);
00542         blist->count--;
00543         blist->version[bucket]++;
00544         objunlock(blist);
00545     }
00546     pthread_mutex_unlock(&blist->locks[bucket]);
00547 }
00548
00552 extern int bucket_list_cnt(struct bucket_list *blist) {
00553     int ret = -1;
00554
00555     if (blist) {
00556         objlock(blist);
00557         ret = blist->count;
00558         objunlock(blist);
00559     }
00560     return (ret);
00561 }
00562
00572 extern void *bucket_list_find_key(struct bucket_list *blist, const void *key
      ) {
00573     struct blist_obj *entry;
00574     int hash, bucket;
00575
00576     if (!blist) {
00577         return (NULL);
00578     }
00579
00580     hash = gethash(blist, key, 1);
00581     bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
      bucketbits) - 1));
00582
00583     pthread_mutex_lock(&blist->locks[bucket]);
00584     entry = blist_gotohash(blist->list[bucket], hash + 1, blist->bucketbits);
00585     if (entry && entry->data) {
00586         objref(entry->data->data);
00587     } else
00588         if (!entry) {
00589             pthread_mutex_unlock(&blist->locks[bucket]);
00590             return NULL;
00591         }
00592
00593     pthread_mutex_unlock(&blist->locks[bucket]);
00594
00595     if (entry->data && (entry->hash == hash)) {
00596         return (entry->data->data);
00597     } else
00598         if (entry->data) {
00599             objunref(entry->data->data);
00600         }
00601
00602     return NULL;
00603 }
00604
00613 extern void bucketlist_callback(struct bucket_list *blist,
      blist_cb callback, void *data2) {
00614     struct bucket_loop *bloop;
00615     void *data;
00616
00617     if (!blist || !callback) {
00618         return;
00619     }
00620
00621     bloop = init_bucket_loop(blist);
00622     while(blist && bloop && (data = next_bucket_loop(bloop))) {
00623         callback(data, data2);
00624         objunref(data);
00625     }
00626     objunref(bloop);
00627 }
00628
00629 static void free_bloop(void *data) {
00630     struct bucket_loop *bloop = data;
```

```
00631
00632     if (bloop->blist) {
00633         objunref(bloop->blist);
00634     }
00635 }
00636
00640 extern struct bucket_loop *init_bucket_loop(struct
      bucket_list *blist) {
00641     struct bucket_loop *bloop = NULL;
00642
00643     if (blist && (bloop = objalloc(sizeof(*bloop), free_bloop))) {
00644         objref(blist);
00645         bloop->blist = blist;
00646         bloop->bucket = 0;
00647         pthread_mutex_lock(&blist->locks[bloop->bucket]);
00648         bloop->head = blist->list[0];
00649         if (bloop->head) {
00650             bloop->head_hash = bloop->head->hash;
00651         };
00652         bloop->version = blist->version[0];
00653         pthread_mutex_unlock(&blist->locks[bloop->bucket]);
00654     }
00655
00656     return (bloop);
00657 }
00658
00662 extern void *next_bucket_loop(struct bucket_loop *bloop) {
00663     struct bucket_list *blist = bloop->blist;
00664     struct ref_obj *entry = NULL;
00665     void *data = NULL;
00666
00667     pthread_mutex_lock(&blist->locks[bloop->bucket]);
00668     if (bloop->head_hash && (blist->version[bloop->bucket] != bloop->
      version)) {
00669         /* bucket has changed unexpectedly i need to ff/rew to hash*/
00670         bloop->head = blist_gotohash(blist->list[bloop->bucket], bloop->
      head_hash + 1, blist->bucketbits);
00671         /*if head has gone find next suitable ignore any added*/
00672         while (bloop->head && (bloop->head->hash < bloop->head_hash)) {
00673             bloop->head = bloop->head->next;
00674         }
00675     }
00676
00677     while (!bloop->head || !bloop->head->prev) {
00678         pthread_mutex_unlock(&blist->locks[bloop->bucket]);
00679         bloop->bucket++;
00680         if (bloop->bucket < (1 << blist->bucketbits)) {
00681             pthread_mutex_lock(&blist->locks[bloop->bucket]);
00682             bloop->head = blist->list[bloop->bucket];
00683         } else {
00684             return NULL;
00685         }
00686     }
00687
00688     if (bloop->head) {
00689         bloop->cur = bloop->head;
00690         entry = (bloop->head->data) ? bloop->head->data : NULL;
00691         data = (entry) ? entry->data : NULL;
00692         objref(data);
00693         bloop->head = bloop->head->next;
00694         bloop->head_hash = (bloop->head) ? bloop->head->hash : 0;
00695         bloop->cur_hash = (bloop->cur) ? bloop->cur->hash : 0;
00696     }
00697     pthread_mutex_unlock(&blist->locks[bloop->bucket]);
00698
00699     return (data);
00700 }
00701
00702
00710 extern void remove_bucket_loop(struct bucket_loop *bloop) {
00711     struct bucket_list *blist = bloop->blist;
00712     int bucket = bloop->bucket;
00713
00714     pthread_mutex_lock(&blist->locks[bloop->bucket]);
00715     /*if the bucket has altered need to verify i can remove*/
00716     if (bloop->cur_hash && (!bloop->cur || (blist->version[bloop->
      bucket] != bloop->version))) {
00717         bloop->cur = blist_gotohash(blist->list[bloop->bucket], bloop->
      cur_hash + 1, blist->bucketbits);
00718         if (!bloop->cur || (bloop->cur->hash != bloop->cur_hash)) {
00719             pthread_mutex_unlock(&blist->locks[bucket]);
00720             return;
00721         }
00722     }
00723
00724     if (!bloop->cur) {
00725         pthread_mutex_unlock(&blist->locks[bucket]);
```

```
00726         return;
00727     }
00728
00729     if (bloop->cur->next && (bloop->cur == blist->list[bucket])) {
00730         bloop->cur->next->prev = bloop->cur->prev;
00731         blist->list[bucket] = bloop->cur->next;
00732     } else if (bloop->cur->next) {
00733         bloop->cur->next->prev = bloop->cur->prev;
00734         bloop->cur->prev->next = bloop->cur->next;
00735     } else if (bloop->cur == blist->list[bucket]) {
00736         blist->list[bucket] = NULL;
00737     } else {
00738         bloop->cur->prev->next = NULL;
00739         blist->list[bucket]->prev = bloop->cur->prev;
00740     }
00741
00742     objunref(bloop->cur->data->data);
00743     free(bloop->cur);
00744     bloop->cur_hash = 0;
00745     bloop->cur = NULL;
00746     blist->version[bucket]++;
00747     bloop->version++;
00748     pthread_mutex_unlock(&blist->locks[bucket]);
00749
00750     objlock(blist);
00751     blist->count--;
00752     objunlock(blist);
00753 }
00754
```

14.44 src/rfc6296.c File Reference

Implementation of RFC6296.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include "include/dtsapp.h"
```

Data Structures

- struct natmap

  RFC6296 Nat map.

Functions

- void rfc6296_map (struct natmap *map, struct in6_addr *ipaddr, int out)

  Lookup and process a NAT transform as per RFC 6296.
- int rfc6296_map_add (char *intaddr, char *extaddr)

  Calculate and add a NAT map.
- void rfc6296_test (blist_cb callback, struct in6_addr *internal)

  Quick test function.

14.44.1 Detailed Description

Implementation of RFC6296.

Definition in file rfc6296.c.

14.45 rfc6296.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00025 #include <stdio.h>
00026 #include <stdlib.h>
00027 #include <string.h>
00028 #include <netinet/in.h>
00029
00030 #include "include/dtsapp.h"
00031
00033 struct natmap {
00035     uint16_t mask;
00037     uint16_t adjo;
00039     uint16_t adji;
00041     uint8_t ipre[16];
00043     uint8_t epre[16];
00044 };
00045
00046 static struct bucket_list *nptv6tbl = NULL;
00047
00048 static int32_t nptv6_hash(const void *data, int key) {
00049     const struct natmap *map = data;
00050     const void *hashkey = (key) ? data : map->ipre;
00051     int ret;
00052
00053     ret = jenhash(hashkey, sizeof(map->ipre), 0);
00054
00055     return (ret);
00056 }
00057
00062 extern void rfc6296_map(struct natmap *map, struct in6_addr *ipaddr, int out) {
00063     uint16_t *addr_16 = (uint16_t *)&ipaddr->s6_addr;
00064     uint32_t calc;
00065     uint8_t cnt, *prefix, bitlen, bytelen;
00066     uint16_t adj;
00067
00068     prefix = (out) ? map->epre : map->ipre;
00069     adj = (out) ? map->adjo : map->adji;
00070
00071     if ((bitlen = map->mask % 8)) {
00072         bytelen = (map->mask - bitlen) / 8;
00073         bytelen++;
00074     } else {
00075         bytelen = map->mask / 8;
00076     }
00077
00078     /*as per RFC we handle /48 and longer /48 changes are reflected in SN*/
00079     if ((bytelen == 6) && (~addr_16[3]) && (!bitlen)) {
00080         memcpy(&ipaddr->s6_addr, prefix, bytelen);
00081         calc = ntohs(addr_16[3]) + adj;
00082         addr_16[3] = htons((calc & 0xFFFF) + (calc >> 16));
00083         if (! ~addr_16[3]) {
00084             addr_16[3] = 0;
00085         }
00086     } else if ((bytelen > 6) && (bytelen < 15)) {
00087         /* find first non 0xFFFF word in lower 64 bits*/
00088         for(cnt = ((bytelen-1) >> 1) + 1; cnt < 8; cnt++) {
00089             if (! ~addr_16[cnt]) {
00090                 continue;
00091             }
00092             if (bitlen) {
00093                 ipaddr->s6_addr[bytelen-1] = prefix[bytelen-1] | (ipaddr->s6_addr[bytelen-1] & ((1 << (8 -
      bitlen)) -1));
00094             } else {
00095                 ipaddr->s6_addr[bytelen-1] = prefix[bytelen-1];
00096             }
00097             memcpy(&ipaddr->s6_addr, prefix, bytelen - 1);
00098             calc = ntohs(addr_16[cnt]) + adj;
00099             addr_16[cnt] = htons((calc & 0xFFFF) + (calc >> 16));
```

```
00100             if (! ~addr_16[cnt]) {
00101                 addr_16[cnt] = 0;
00102             }
00103             break;
00104         }
00105     }
00106 }
00107
00111 extern int rfc6296_map_add(char *intaddr, char *extaddr) {
00112     struct natmap *map;
00113     uint16_t emask, imask, isum, esum, bytelen, bitlen;
00114     char inip[43], exip[43], *tmp2;
00115     struct in6_addr i6addr;
00116     uint32_t adj;
00117
00118     strncpy(inip, intaddr, 43);
00119     if ((tmp2 = rindex(inip, '/'))) {
00120         tmp2[0] = '\0';
00121         tmp2++;
00122         imask = atoi(tmp2);
00123     } else {
00124         return (-1);
00125     }
00126
00127     strncpy(exip, extaddr, 43);
00128     if ((tmp2 = rindex(exip, '/'))) {
00129         tmp2[0] = '\0';
00130         tmp2++;
00131         emask = atoi(tmp2);
00132     } else {
00133         return (-1);
00134     }
00135
00136     map = objalloc(sizeof(*map), NULL);
00137     map->mask = (emask > imask) ? emask : imask;
00138
00139     /*rfc says we must zero extend this is what we do here looking at each supplied len*/
00140     /*external range*/
00141     inet_pton(AF_INET6, exip, &i6addr);
00142     if ((bitlen = emask % 8)) {
00143         bytelen = (emask - bitlen) / 8;
00144         i6addr.s6_addr[bytelen] &= ~((1 << (8 - bitlen)) - 1);
00145         bytelen++;
00146     } else {
00147         bytelen = emask / 8;
00148     }
00149     memcpy(map->epre, &i6addr.s6_addr, bytelen);
00150
00151     /*internal range*/
00152     inet_pton(AF_INET6, inip, &i6addr);
00153     if ((bitlen = imask % 8)) {
00154         bytelen = (imask - bitlen) / 8;
00155         i6addr.s6_addr[bytelen] &= ~((1 << (8 - bitlen)) - 1);
00156         bytelen++;
00157     } else {
00158         bytelen = imask / 8;
00159     }
00160     memcpy(map->ipre, &i6addr.s6_addr, bytelen);
00161
00162     /*calculate the adjustments from checksums of prefixes*/
00163     if ((bitlen = map->mask % 8)) {
00164         bytelen = (map->mask - bitlen) / 8;
00165         bytelen++;
00166     } else {
00167         bytelen = map->mask / 8;
00168     }
00169     esum = ntohs(checksum(map->epre, bytelen));
00170     isum = ntohs(checksum(map->ipre, bytelen));
00171
00172     /*outgoing transform*/
00173     adj = esum - isum;
00174     adj = (adj & 0xFFFF) + (adj >> 16);
00175     map->adjo = (uint16_t)adj;
00176
00177     /*incoming transform*/
00178     adj = isum - esum;
00179     adj = (adj & 0xFFFF) + (adj >> 16);
00180     map->adji = (uint16_t)adj;
00181
00182     if (!nptv6tbl && (!(nptv6tbl = create_bucketlist(5, nptv6_hash)))) {
00183         objunref(map);
00184         return (-1);
00185     }
00186     addtobucket(nptv6tbl, map);
00187     objunref(map);
00188
00189     return (0);
```

```
00190 }
00191
00197 extern void rfc6296_test(blist_cb callback, struct in6_addr *internal) {
00198     /*find and run map*/
00199     bucketlist_callback(nptv6tbl, callback, internal);
00200
00201     objunref(nptv6tbl);
00202 }
00203
```

14.46 src/sslutil.c File Reference

TLSv1 SSLv2 SSLv3 DTLSv1 support.

```
#include <stdint.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "include/dtsapp.h"
```

Data Structures

- struct ssldata

  SSL data structure for enabling encryption on sockets.

Macros

- #define COOKIE_SECRET_LENGTH 32

  length of cookie secret using SHA2-256 HMAC

Enumerations

- enum SSLFLAGS {
  SSL_TLSV1 = 1 << 0, SSL_SSLV2 = 1 << 1, SSL_SSLV3 = 1 << 2, SSL_DTLSV1 = 1 << 3,
  SSL_CLIENT = 1 << 4, SSL_SERVER = 1 << 5, SSL_DTLSCON = 1 << 6 }

  SSL configuration flags.

Functions

- void ssl_shutdown (void *data, int sock)

  Shutdown the SSL connection.
- void * tlsv1_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for TLSv1.
- void * sslv2_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for SSLv2 (If available)
- void * sslv3_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for SSLv3.
- void * dtlsv1_init (const char *cacert, const char *cert, const char *key, int verify)

  Create a SSL structure for DTLSv1.
- void tlsaccept (struct fwsocket *sock, struct ssldata *orig)

Create SSL session for new connection.

- int socketread_d (struct fwsocket *sock, void *buf, int num, union sockstruct *addr)

  Read from a socket into a buffer.
- int socketread (struct fwsocket *sock, void *buf, int num)

  Read from a socket into a buffer.
- int socketwrite_d (struct fwsocket *sock, const void *buf, int num, union sockstruct *addr)

  Write a buffer to a socket.
- int socketwrite (struct fwsocket *sock, const void *buf, int num)

  Write a buffer to a socket.
- void sslstartup (void)

  Initialise SSL support this should be called at startup.
- void dtsl_serveropts (struct fwsocket *sock)

  Start up the DTLSv1 Server.
- struct fwsocket * dtls_listenssl (struct fwsocket *sock)

  Implementation of "listen" for DTLSv1.
- void startsslclient (struct fwsocket *sock)

  Start SSL on a client socket.
- void dtlstimeout (struct fwsocket *sock, struct timeval *timeleft, int defusec)

  Get DTLSv1 timeout setting todefault timeout.
- void dtlshandltimeout (struct fwsocket *sock)

  Handle DTLSv1 timeout.

14.46.1 Detailed Description

TLSv1 SSLv2 SSLv3 DTLSv1 support.

Definition in file sslutil.c.

14.47 sslutil.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00030 #include <stdint.h>
00031 #ifdef __WIN32__
00032 #include <winsock2.h>
00033 #include <windows.h>
00034 #include <ws2tcpip.h>
00035 #endif
00036 #include <openssl/ssl.h>
00037 #include <openssl/err.h>
00038 #include <sys/stat.h>
00039 #include <unistd.h>
00040 #ifndef __WIN32__
00041 #include <sys/socket.h>
00042 #include <arpa/inet.h>
00043 #endif
00044
00045 #include "include/dtsapp.h"
00046
```

```
00048 enum SSLFLAGS {
00050     SSL_TLSV1  = 1 << 0,
00052     SSL_SSLV2  = 1 << 1,
00054     SSL_SSLV3  = 1 << 2,
00056     SSL_DTLSV1    = 1 << 3,
00058     SSL_CLIENT    = 1 << 4,
00060     SSL_SERVER    = 1 << 5,
00062     SSL_DTLSCON  = 1 << 6
00063 };
00064
00066 struct ssldata {
00068     SSL_CTX *ctx;
00070     SSL *ssl;
00072     BIO *bio;
00075     int flags;
00077     const SSL_METHOD *meth;
00079     struct ssldata *parent;
00080 };
00081
00083 #define COOKIE_SECRET_LENGTH 32
00084 static unsigned char *cookie_secret = NULL;
00085
00086 static int generate_cookie(SSL *ssl, unsigned char *cookie, unsigned int *cookie_len) {
00087     union sockstruct peer;
00088
00089     if (!ssl || !cookie_secret || (*cookie_len < COOKIE_SECRET_LENGTH)) {
00090         return (0);
00091     }
00092
00093     memset(&peer, 0, sizeof(peer));
00094     BIO_dgram_get_peer(SSL_get_rbio(ssl), &peer);
00095     sha256hmac(cookie, &peer, sizeof(peer), cookie_secret,
      COOKIE_SECRET_LENGTH);
00096     *cookie_len = COOKIE_SECRET_LENGTH;
00097
00098     return (1);
00099 }
00100
00101 static int verify_cookie(SSL *ssl, unsigned char *cookie, unsigned int cookie_len) {
00102     union sockstruct peer;
00103     unsigned char hmac[COOKIE_SECRET_LENGTH];
00104
00105     if (!ssl || !cookie_secret || (cookie_len != COOKIE_SECRET_LENGTH)) {
00106         return (0);
00107     }
00108
00109     memset(&peer, 0, sizeof(peer));
00110     BIO_dgram_get_peer(SSL_get_rbio(ssl), &peer);
00111     sha256hmac(hmac, &peer, sizeof(peer), cookie_secret,
      COOKIE_SECRET_LENGTH);
00112
00113     if (!sha256cmp(hmac, cookie)) {
00114         return (1);
00115     }
00116
00117     return (0);
00118 }
00119
00120 static int _ssl_shutdown(struct ssldata *ssl) {
00121     int err, ret = 0;
00122
00123     if ((ret = SSL_shutdown(ssl->ssl)) < 1) {
00124         objunlock(ssl);
00125         if (ret == 0) {
00126             objlock(ssl);
00127                 ret = SSL_shutdown(ssl->ssl);
00128         } else {
00129             objlock(ssl);
00130         }
00131         err = SSL_get_error(ssl->ssl, ret);
00132         switch(err) {
00133             case SSL_ERROR_WANT_READ:
00134                 ret = 1;
00135                 break;
00136             case SSL_ERROR_WANT_WRITE:
00137                 ret = -1;
00138                 break;
00139             case SSL_ERROR_SSL:
00140                 /*ignore im going away now*/
00141             case SSL_ERROR_SYSCALL:
00142                 /* ignore this as documented*/
00143             case SSL_ERROR_NONE:
00144                 /* nothing to see here moving on*/
00145                 break;
00146             default:
00147                 printf("SSL Shutdown unknown error %i\n", err);
00148                 break;
```

```
00149         }
00150     }
00151     return ret;
00152 }
00153 
00154 
00155 static int socket_select(int sock, int read) {
00156     int selfd;
00157     struct timeval tv;
00158     fd_set act_set;
00159     FD_ZERO(&act_set);
00160     FD_SET(sock, &act_set);
00161     tv.tv_sec = 0;
00162     tv.tv_usec = 100000;
00163 
00164     if (read == 1) {
00165         selfd = select(sock + 1, &act_set, NULL, NULL, &tv);
00166     } else {
00167         selfd = select(sock + 1, NULL, &act_set, NULL, &tv);
00168     }
00169     return selfd;
00170 }
00171 
00179 extern void ssl_shutdown(void *data, int sock) {
00180     struct ssldata *ssl = data;
00181     int ret, selfd, cnt = 0;
00182 
00183     if (!ssl) {
00184         return;
00185     }
00186 
00187     objlock(ssl);
00188 
00189     while (ssl->ssl &&  (ret = _ssl_shutdown(ssl) && (cnt < 3))) {
00190         selfd = socket_select(sock, ret);
00191         if (selfd <= 0) {
00192             break;
00193         }
00194         cnt++;
00195     }
00196 
00197     if (ssl->ssl) {
00198         SSL_free(ssl->ssl);
00199         ssl->ssl = NULL;
00200     }
00201     objunlock(ssl);
00202 }
00203 
00204 static void free_ssldata(void *data) {
00205     struct ssldata *ssl = data;
00206 
00207     if (ssl->parent) {
00208         objunref(ssl->parent);
00209     }
00210 
00211     if (ssl->ctx) {
00212         SSL_CTX_free(ssl->ctx);
00213         ssl->ctx = NULL;
00214     }
00215 }
00216 
00217 static int verify_callback (int ok, X509_STORE_CTX *ctx) {
00218     return (1);
00219 }
00220 
00221 static struct ssldata *sslinit(const char *cacert, const char *cert, const char *key, int verify,
    const SSL_METHOD *meth, int flags) {
00222     struct ssldata *ssl;
00223     struct stat finfo;
00224     int ret = -1;
00225 
00226     if (!(ssl = objalloc(sizeof(*ssl), free_ssldata))) {
00227         return NULL;
00228     }
00229 
00230     ssl->flags = flags;
00231     ssl->meth = meth;
00232     if (!(ssl->ctx = SSL_CTX_new(meth))) {
00233         objunref(ssl);
00234         return NULL;
00235     }
00236 
00237     if (!stat(cacert, &finfo)) {
00238         if (S_ISDIR(finfo.st_mode) && (SSL_CTX_load_verify_locations(ssl->ctx, NULL, cacert) == 1)) {
00239             ret = 0;
00240         } else
00241             if (SSL_CTX_load_verify_locations(ssl->ctx, cacert, NULL) == 1) {
```

```
00242                 ret = 0;
00243             }
00244     }
00245
00246     if (!ret && (SSL_CTX_use_certificate_file(ssl->ctx, cert, SSL_FILETYPE_PEM) == 1)) {
00247         ret = 0;
00248     }
00249     if (!ret && (SSL_CTX_use_PrivateKey_file(ssl->ctx, key, SSL_FILETYPE_PEM) == 1)) {
00250         ret = 0;
00251     }
00252
00253     if (!ret && (SSL_CTX_check_private_key (ssl->ctx) == 1)) {
00254         ret= 0;
00255     }
00256
00257     /*XXX   Should create a tmp 512 bit rsa key for RSA ciphers also need DH
00258         http://www.openssl.org/docs/ssl/SSL_CTX_set_cipher_list.html
00259         SSL_CTX_set_cipher_list*/
00260
00261     if (!ret) {
00262         /* XXX CRL verification
00263                 X509_VERIFY_PARAM *param;
00264                 param = X509_VERIFY_PARAM_new();
00265                 X509_VERIFY_PARAM_set_flags(param, X509_V_FLAG_CRL_CHECK);
00266                 SSL_CTX_set1_param(ctx, param);
00267                 X509_VERIFY_PARAM_free(param);
00268         */
00269         SSL_CTX_set_verify(ssl->ctx, verify, verify_callback);
00270         SSL_CTX_set_verify_depth(ssl->ctx, 1);
00271     }
00272
00273     if (ret) {
00274         objunref(ssl);
00275         return NULL;
00276     }
00277
00278     return (ssl);
00279 }
00280
00281
00287 extern void *tlsv1_init(const char *cacert, const char *cert, const char *key, int verify) {
00288     const SSL_METHOD *meth = TLSv1_method();
00289
00290     return (sslinit(cacert, cert, key, verify, meth, SSL_TLSV1));
00291 }
00292
00298 #ifndef OPENSSL_NO_SSL2
00299 extern void *sslv2_init(const char *cacert, const char *cert, const char *key, int verify) {
00300     const SSL_METHOD *meth = SSLv2_method();
00301
00302     return (sslinit(cacert, cert, key, verify, meth, SSL_SSLV2));
00303 }
00304 #endif
00305
00311 extern void *sslv3_init(const char *cacert, const char *cert, const char *key, int verify) {
00312     const SSL_METHOD *meth = SSLv3_method();
00313     struct ssldata *ssl;
00314
00315     ssl = sslinit(cacert, cert, key, verify, meth, SSL_SSLV3);
00316
00317     return (ssl);
00318 }
00319
00325 extern void *dtlsv1_init(const char *cacert, const char *cert, const char *key, int verify) {
00326     const SSL_METHOD *meth = DTLSv1_method();
00327     struct ssldata *ssl;
00328
00329     ssl = sslinit(cacert, cert, key, verify, meth, SSL_DTLSV1);
00330     /* XXX BIO_CTRL_DGRAM_MTU_DISCOVER*/
00331     SSL_CTX_set_read_ahead(ssl->ctx, 1);
00332
00333     return (ssl);
00334 }
00335
00336 static void sslsockstart(struct fwsocket *sock, struct ssldata *orig,int accept) {
00337     struct ssldata *ssl = sock->ssl;
00338
00339     if (!ssl) {
00340         return;
00341     }
00342
00343     objlock(sock);
00344     objlock(ssl);
00345     if (orig) {
00346         objlock(orig);
00347         ssl->ssl = SSL_new(orig->ctx);
00348         objunlock(orig);
```

```
00349     } else {
00350         ssl->ssl = SSL_new(ssl->ctx);
00351     }
00352 
00353     if (ssl->ssl) {
00354         ssl->bio = BIO_new_socket(sock->sock, BIO_NOCLOSE);
00355         objunlock(sock);
00356         SSL_set_bio(ssl->ssl, ssl->bio, ssl->bio);
00357         if (accept) {
00358             SSL_accept(ssl->ssl);
00359             ssl->flags |= SSL_SERVER;
00360         } else {
00361             SSL_connect(ssl->ssl);
00362             ssl->flags |= SSL_CLIENT;
00363         }
00364         if (orig) {
00365             objref(orig);
00366             ssl->parent = orig;
00367         }
00368         objunlock(ssl);
00369     } else {
00370         objunlock(ssl);
00371         objunref(ssl);
00372         sock->ssl = NULL;
00373         objunlock(sock);
00374         return;
00375     }
00376 }
00377 
00382 extern void tlsaccept(struct fwsocket *sock, struct ssldata *orig) {
00383     setflag(sock, SOCK_FLAG_SSL);
00384     if ((sock->ssl = objalloc(sizeof(*sock->ssl), free_ssldata))) {
00385         sslsockstart(sock, orig, 1);
00386     }
00387 }
00388 
00406 extern int socketread_d(struct fwsocket *sock, void *buf, int num, union
      sockstruct *addr) {
00407     struct ssldata *ssl = sock->ssl;
00408     socklen_t salen = sizeof(*addr);
00409     int ret, err, syserr;
00410 
00411     if (!ssl && !testflag(sock, SOCK_FLAG_SSL)) {
00412         objlock(sock);
00413         if (addr && (sock->type == SOCK_DGRAM)) {
00414             ret = recvfrom(sock->sock, buf, num, 0, &addr->sa, &salen);
00415         } else {
00416 #ifndef __WIN32
00417             ret = read(sock->sock, buf, num);
00418 #else
00419             ret = recv(sock->sock, buf, num, 0);
00420 #endif
00421         }
00422         if (ret == 0) {
00423             sock->flags |= SOCK_FLAG_CLOSE;
00424         }
00425         objunlock(sock);
00426         return (ret);
00427     } else if (!ssl) {
00428         return -1;
00429     }
00430 
00431     objlock(ssl);
00432     /* ive been shutdown*/
00433     if (!ssl->ssl) {
00434         objunlock(ssl);
00435         return (-1);
00436     }
00437     ret = SSL_read(ssl->ssl, buf, num);
00438     err = SSL_get_error(ssl->ssl, ret);
00439     if (ret == 0) {
00440         sock->flags |= SOCK_FLAG_CLOSE;
00441     }
00442     objunlock(ssl);
00443     switch (err) {
00444         case SSL_ERROR_NONE:
00445             break;
00446         case SSL_ERROR_WANT_X509_LOOKUP:
00447             printf("Want X509\n");
00448             break;
00449         case SSL_ERROR_WANT_READ:
00450             printf("Read Want Read\n");
00451             break;
00452         case SSL_ERROR_WANT_WRITE:
00453             printf("Read Want write\n");
00454             break;
00455         case SSL_ERROR_ZERO_RETURN:
```

```
00456         case SSL_ERROR_SSL:
00457             objlock(sock);
00458             objunref(sock->ssl);
00459             sock->ssl = NULL;
00460             objunlock(sock);
00461             break;
00462         case SSL_ERROR_SYSCALL:
00463             syserr = ERR_get_error();
00464             if (syserr || (!syserr && (ret == -1))) {
00465                 printf("R syscall %i %i\n", syserr, ret);
00466             }
00467             break;
00468         default
00469                 :
00470             printf("other\n");
00471             break;
00472     }
00473
00474     return (ret);
00475 }
00476
00489 extern int socketread(struct fwsocket *sock, void *buf, int num) {
00490     return (socketread_d(sock, buf, num, NULL));
00491 }
00492
00493
00508 extern int socketwrite_d(struct fwsocket *sock, const void *buf, int num, union
      sockstruct *addr) {
00509     struct ssldata *ssl = (sock) ? sock->ssl : NULL;
00510     int ret, err, syserr;
00511
00512     if (!sock) {
00513         return (-1);
00514     }
00515
00516     if (!ssl && !testflag(sock, SOCK_FLAG_SSL)) {
00517         objlock(sock);
00518         if (addr && (sock->type == SOCK_DGRAM)) {
00519 #ifndef __WIN32
00520             if (sock->flags & SOCK_FLAG_UNIX) {
00521                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, (const struct sockaddr *)&addr->
      un, sizeof(addr->un));
00522             } else if (sock->flags & SOCK_FLAG_MCAST) {
00523                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &sock->addr.
      sa, sizeof(sock->addr.ss));
00524             } else {
00525                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &addr->sa, sizeof(*addr));
00526             }
00527 #else
00528             if (sock->flags & SOCK_FLAG_MCAST) {
00529                 ret = sendto(sock->sock, buf, num, 0, &sock->addr.sa, sizeof(sock->
      addr.ss));
00530             } else {
00531                 ret = sendto(sock->sock, buf, num, 0, &addr->sa, sizeof(*addr));
00532             }
00533 #endif
00534         } else {
00535 #ifndef __WIN32
00536             if (sock->flags & SOCK_FLAG_MCAST) {
00537                 ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &sock->addr.
      sa, sizeof(sock->addr.ss));
00538             } else {
00539                 ret = send(sock->sock, buf, num, MSG_NOSIGNAL);
00540             }
00541 #else
00542             if (sock->flags & SOCK_FLAG_MCAST) {
00543                 ret = sendto(sock->sock, buf, num, 0, &sock->addr.sa, sizeof(sock->
      addr.ss));
00544             } else {
00545                 ret = send(sock->sock, buf, num, 0);
00546             }
00547 #endif
00548         }
00549         if (ret == -1) {
00550             switch(errno) {
00551                 case EBADF:
00552                 case EPIPE:
00553 #ifndef __WIN32
00554                 case ENOTCONN:
00555                 case ENOTSOCK:
00556 #endif
00557                     sock->flags |= SOCK_FLAG_CLOSE;
00558                     break;
00559             }
00560         }
00561         objunlock(sock);
00562         return (ret);
```

```
00563     } else if (!ssl) {
00564         return -1;
00565     }
00566
00567     if (ssl && ssl->ssl) {
00568         objlock(ssl);
00569         if (SSL_state(ssl->ssl) != SSL_ST_OK) {
00570             objunlock(ssl);
00571             return (SSL_ERROR_SSL);
00572         }
00573         ret = SSL_write(ssl->ssl, buf, num);
00574         err = SSL_get_error(ssl->ssl, ret);
00575         objunlock(ssl);
00576     } else {
00577         return -1;
00578     }
00579
00580     if (ret == -1) {
00581         setflag(sock, SOCK_FLAG_CLOSE);
00582     }
00583
00584     switch(err) {
00585         case SSL_ERROR_NONE:
00586             break;
00587         case SSL_ERROR_WANT_READ:
00588             printf("Send Want Read\n");
00589             break;
00590         case SSL_ERROR_WANT_WRITE:
00591             printf("Send Want write\n");
00592             break;
00593         case SSL_ERROR_WANT_X509_LOOKUP:
00594             printf("Want X509\n");
00595             break;
00596         case SSL_ERROR_ZERO_RETURN:
00597         case SSL_ERROR_SSL:
00598             objlock(sock);
00599             objunref(sock->ssl);
00600             sock->ssl = NULL;
00601             objunlock(sock);
00602             break;
00603         case SSL_ERROR_SYSCALL:
00604             syserr = ERR_get_error();
00605             if (syserr || (!syserr && (ret == -1))) {
00606                 printf("W syscall %i %i\n", syserr, ret);
00607             }
00608             break;
00609         default:
00610             printf("other\n");
00611             break;
00612     }
00613
00614     return (ret);
00615 }
00616
00629 extern int socketwrite(struct fwsocket *sock, const void *buf, int num) {
00630     return (socketwrite_d(sock, buf, num, NULL));
00631 }
00632
00639 extern void sslstartup(void) {
00640     SSL_library_init();
00641     SSL_load_error_strings();
00642     OpenSSL_add_ssl_algorithms();
00643
00644     if ((cookie_secret = malloc(COOKIE_SECRET_LENGTH))) {
00645         genrand(cookie_secret, COOKIE_SECRET_LENGTH);
00646     }
00647 }
00648
00649 static void dtlssetopts(struct ssldata *ssl, struct ssldata *orig, struct
      fwsocket *sock) {
00650     struct timeval timeout;
00651
00652     objlock(sock);
00653     objlock(ssl);
00654     ssl->bio = BIO_new_dgram(sock->sock, BIO_NOCLOSE);
00655     objunlock(sock);
00656
00657     timeout.tv_sec = 5;
00658     timeout.tv_usec = 0;
00659     BIO_ctrl(ssl->bio, BIO_CTRL_DGRAM_SET_RECV_TIMEOUT, 0, &timeout);
00660     timeout.tv_sec = 5;
00661     timeout.tv_usec = 0;
00662     BIO_ctrl(ssl->bio, BIO_CTRL_DGRAM_SET_SEND_TIMEOUT, 0, &timeout);
00663
00664     if (orig) {
00665         objlock(orig);
00666         if ((ssl->ssl = SSL_new(orig->ctx))) {
```

```
00667             objunlock(orig);
00668             objref(orig);
00669             ssl->parent = orig;
00670         } else {
00671             objunlock(orig);
00672         }
00673     } else {
00674         ssl->ssl = SSL_new(ssl->ctx);
00675     }
00676     SSL_set_bio(ssl->ssl, ssl->bio, ssl->bio);
00677     objunlock(ssl);
00678     setflag(sock, SOCK_FLAG_SSL);
00679 }
00680
00685 extern void dtsl_serveropts(struct fwsocket *sock) {
00686     struct ssldata *ssl = sock->ssl;
00687
00688     if (!ssl) {
00689         return;
00690     }
00691
00692     dtlssetopts(ssl, NULL, sock);
00693
00694     objlock(ssl);
00695     SSL_CTX_set_cookie_generate_cb(ssl->ctx, generate_cookie);
00696     SSL_CTX_set_cookie_verify_cb(ssl->ctx, verify_cookie);
00697     SSL_CTX_set_session_cache_mode(ssl->ctx, SSL_SESS_CACHE_OFF);
00698
00699     SSL_set_options(ssl->ssl, SSL_OP_COOKIE_EXCHANGE);
00700     ssl->flags |= SSL_SERVER;
00701     objunlock(ssl);
00702 }
00703
00704 static void dtlsaccept(struct fwsocket *sock) {
00705     struct ssldata *ssl = sock->ssl;
00706
00707     objlock(sock);
00708     objlock(ssl);
00709     ssl->flags |= SSL_SERVER;
00710
00711     BIO_set_fd(ssl->bio, sock->sock, BIO_NOCLOSE);
00712     BIO_ctrl(ssl->bio, BIO_CTRL_DGRAM_SET_CONNECTED, 0, &sock->addr);
00713     objunlock(sock);
00714
00715     SSL_accept(ssl->ssl);
00716
00717     if (SSL_get_peer_certificate(ssl->ssl)) {
00718         printf ("A------------------------------------------------------------\n");
00719         X509_NAME_print_ex_fp(stdout, X509_get_subject_name(SSL_get_peer_certificate(ssl->
      ssl)), 1, XN_FLAG_MULTILINE);
00720         printf("\n\n Cipher: %s", SSL_CIPHER_get_name(SSL_get_current_cipher(ssl->
      ssl)));
00721         printf ("\n------------------------------------------------------------\n\n");
00722     }
00723     objunlock(ssl);
00724 }
00725
00726
00731 extern struct fwsocket *dtls_listenssl(struct fwsocket *sock) {
00732     struct ssldata *ssl = sock->ssl;
00733     struct ssldata *newssl;
00734     struct fwsocket *newsock;
00735     union sockstruct client;
00736 #ifndef __WIN32__
00737     int on = 1;
00738 #else
00739 /*  unsigned long on = 1;*/
00740 #endif
00741
00742     if (!(newssl = objalloc(sizeof(*newssl), free_ssldata))) {
00743         return NULL;
00744     }
00745
00746     newssl->flags |= SSL_DTLSCON;
00747
00748     dtlssetopts(newssl, ssl, sock);
00749     memset(&client, 0, sizeof(client));
00750     if (DTLSv1_listen(newssl->ssl, &client) <= 0) {
00751         objunref(newssl);
00752         return NULL;
00753     }
00754
00755     objlock(sock);
00756     if (!(newsock = make_socket(sock->addr.sa.sa_family, sock->
      type, sock->proto, newssl))) {
00757         objunlock(sock);
00758         objunref(newssl);
```

```
00759         return NULL;
00760     }
00761     objunlock(sock);
00762     memcpy(&newsock->addr, &client, sizeof(newsock->addr));
00763 #ifndef __WIN32__
00764     setsockopt(newsock->sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
00765 #ifdef SO_REUSEPORT
00766     setsockopt(newsock->sock, SOL_SOCKET, SO_REUSEPORT, &on, sizeof(on));
00767 #endif
00768 #else
00769 /*  ioctlsocket(newsock->sock, FIONBIO, (unsigned long*)&on);*/
00770 #endif
00771     objlock(sock);
00772     bind(newsock->sock, &sock->addr.sa, sizeof(sock->addr));
00773     objunlock(sock);
00774     connect(newsock->sock, &newsock->addr.sa, sizeof(newsock->addr));
00775 
00776     dtlsaccept(newsock);
00777     setflag(newsock, SOCK_FLAG_SSL);
00778 
00779     return (newsock);
00780 }
00781 
00782 static void dtlsconnect(struct fwsocket *sock) {
00783     struct ssldata *ssl = sock->ssl;
00784 
00785     if (!ssl) {
00786         return;
00787     }
00788 
00789     dtlssetopts(ssl, NULL, sock);
00790 
00791     objlock(sock);
00792     objlock(ssl);
00793     ssl->flags |= SSL_CLIENT;
00794     BIO_ctrl(ssl->bio, BIO_CTRL_DGRAM_SET_CONNECTED, 0, &sock->addr);
00795     objunlock(sock);
00796     SSL_connect(ssl->ssl);
00797 
00798     if (SSL_get_peer_certificate(ssl->ssl)) {
00799         printf ("C------------------------------------------------------------\n");
00800         X509_NAME_print_ex_fp(stdout, X509_get_subject_name(SSL_get_peer_certificate(ssl->
      ssl)), 1, XN_FLAG_MULTILINE);
00801         printf("\n\n Cipher: %s", SSL_CIPHER_get_name(SSL_get_current_cipher(ssl->
      ssl)));
00802         printf ("\n------------------------------------------------------------\n\n");
00803     }
00804     objunlock(ssl);
00805 }
00806 
00811 extern void startsslclient(struct fwsocket *sock) {
00812     if (!sock || !sock->ssl || (sock->ssl->flags & SSL_SERVER)) {
00813         return;
00814     }
00815 
00816     switch(sock->type) {
00817         case SOCK_DGRAM:
00818             dtlsconnect(sock);
00819             break;
00820         case SOCK_STREAM:
00821             sslsockstart(sock, NULL, 0);
00822             break;
00823     }
00824 }
00825 
00831 extern void dtlstimeout(struct fwsocket *sock, struct timeval *timeleft, int defusec) {
00832     if (!sock || !sock->ssl || !sock->ssl->ssl) {
00833         return;
00834     }
00835 
00836     objlock(sock->ssl);
00837     if (!DTLSv1_get_timeout(sock->ssl->ssl, timeleft)) {
00838         timeleft->tv_sec = 0;
00839         timeleft->tv_usec = defusec;
00840     }
00841     objunlock(sock->ssl);
00842 }
00843 
00846 extern void dtlshandltimeout(struct fwsocket *sock) {
00847     if (!sock->ssl) {
00848         return;
00849     }
00850 
00851     objlock(sock->ssl);
00852     DTLSv1_handle_timeout(sock->ssl->ssl);
00853     objunlock(sock->ssl);
00854 }
```

00855

14.48 src/thread.c File Reference

Functions for starting and managing threads.

```
#include <pthread.h>
#include <signal.h>
#include <unistd.h>
#include <stdint.h>
#include "include/dtsapp.h"
```

Data Structures

- struct thread_pvt

  thread struct used to create threads data needs to be first element
- struct threadcontainer

  Global threads data.

Enumerations

- enum threadopt {
  TL_THREAD_NONE = 1 << 0, TL_THREAD_RUN = 1 << 1, TL_THREAD_DONE = 1 << 2, TL_THREAD_JOIN = 1 << 3,
  TL_THREAD_STOP = 1 << 4, TL_THREAD_CAN_CANCEL = 1 << 16, TL_THREAD_JOINABLE = 1 << 17, TL_THREAD_RETURN = 1 << 18 }

  Thread status a thread can be disabled by unsetting TL_THREAD_RUN.

Functions

- int framework_threadok ()

  let threads check there status.
- int startthreads (void)

  Initialise the threadlist and start manager thread.
- void stopthreads (int join)

  Signal manager to stop and cancel all running threads.
- struct thread_pvt * framework_mkthread (threadfunc func, threadcleanup cleanup, threadsighandler sig_handler, void *data, int flags)

  create a thread result must be unreferenced
- void jointhreads (void)

  Join the manager thread.
- int thread_signal (int sig)

  Handle signal if its for me.

Variables

- struct threadcontainer * threads = NULL

  Thread control data.
- int thread_can_start = 1

  Automatically start manager thread.

14.48.1 Detailed Description

Functions for starting and managing threads. The thread interface consists of a management thread managing a hashed bucket list of threads running optional clean up when done.

Definition in file thread.c.

14.49 thread.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00027 #include <pthread.h>
00028 #include <signal.h>
00029 #include <unistd.h>
00030 #include <stdint.h>
00031
00032 #include "include/dtsapp.h"
00033
00036 enum threadopt {
00038     TL_THREAD_NONE        = 1 << 0,
00040     TL_THREAD_RUN      = 1 << 1,
00042     TL_THREAD_DONE        = 1 << 2,
00045     TL_THREAD_JOIN        = 1 << 3,
00048     TL_THREAD_STOP        = 1 << 4,
00049
00051     TL_THREAD_CAN_CANCEL    = 1 << 16,
00053     TL_THREAD_JOINABLE    = 1 << 17,
00054     TL_THREAD_RETURN    = 1 << 18
00055 };
00056
00058 struct thread_pvt {
00060     void            *data;
00062     pthread_t       thr;
00065     threadcleanup      cleanup;
00068     threadfunc        func;
00071     threadsighandler    sighandler;
00074     enum                    threadopt flags;
00075 };
00076
00078 struct threadcontainer {
00080     struct bucket_list   *list;
00082     struct thread_pvt *manager;
00083 };
00084
00086 struct threadcontainer *threads = NULL;
00087
00092 int thread_can_start = 1;
00093
00094 static int32_t hash_thread(const void *data, int key) {
00095     const struct thread_pvt *thread = data;
00096     const pthread_t *th = (key) ? data : &thread->thr;
00097     return jenhash(th, sizeof(pthread_t), 0);
00098 }
00099
00100 static void close_threads(void *data) {
00101     struct threadcontainer *tc = data;
00102
00103     if (tc->list) {
00104         objunref(tc->list);
00105     }
00106
00107     if (tc->manager) {
00108         objunref(tc->manager);
00109         tc->manager = NULL;
00110     }
```

```
00111     threads = NULL;
00112 }
00113
00114 static void free_thread(void *data) {
00115     struct thread_pvt *thread = data;
00116
00117     if (thread->data) {
00118         objunref(thread->data);
00119     }
00120 }
00121
00122 static struct thread_pvt *get_thread_from_id() {
00123     struct thread_pvt *thr;
00124     struct threadcontainer *tc;
00125     pthread_t me;
00126
00127     if (!(tc = (objref(threads)) ? threads : NULL)) {
00128         return NULL;
00129     }
00130     me = pthread_self();
00131
00132     objlock(tc);
00133     thr = bucket_list_find_key(tc->list, &me);
00134     objunlock(tc);
00135     objunref(tc);
00136     return thr;
00137 }
00138
00139
00143 extern int framework_threadok() {
00144     struct thread_pvt *thr;
00145     int ret;
00146
00147     thr = get_thread_from_id();
00148     ret =(thr) ? testflag(thr, TL_THREAD_RUN) : 0;
00149     objunref(thr);
00150
00151     return ret;
00152 }
00153
00154 /*
00155  * close all threads when we get SIGHUP
00156  */
00157 static int manager_sig(int sig, void *data) {
00158 #ifndef __WIN32
00159     struct thread_pvt *thread;
00160
00161     if (!(thread = get_thread_from_id())) {
00162         return 0;
00163     }
00164
00165
00166     switch(sig) {
00167         case SIGHUP:
00168             clearflag(thread, TL_THREAD_RUN);
00169             break;
00170         case SIGINT:
00171         case SIGTERM:
00172             clearflag(thread, TL_THREAD_RUN);
00173             setflag(thread, TL_THREAD_STOP);
00174     }
00175     objunref(thread);
00176     return 1;
00177 #else
00178     return 0;
00179 #endif
00180 }
00181
00182 /* if im here im the last thread*/
00183 static void manage_clean(void *data) {
00184
00185     /*make sure im still here when turning off*/
00186     objlock(threads);
00187     thread_can_start = 0;
00188     objunlock(threads);
00189     objunref(threads);
00190 }
00191
00192 static void stop_threads(void *data, void *data2) {
00193     struct thread_pvt *thread = data;
00194     struct thread_pvt *man = data2;
00195
00196     /*Dont footbullet*/
00197     if (!pthread_equal(man->thr, thread->thr)) {
00198         if (thread->sighandler) {
00199             pthread_kill(thread->thr, SIGTERM);
00200         }
```

```
00201         if (testflag(thread, TL_THREAD_CAN_CANCEL) &&
      testflag(thread, TL_THREAD_RUN)) {
00202             pthread_cancel(thread->thr);
00203         }
00204         clearflag(thread, TL_THREAD_RUN);
00205     }
00206 }
00207
00208 /*
00209  * loop through all threads till they stoped
00210  * setting stop will flag threads to stop
00211  */
00212 static void *managethread(void *data) {
00213     struct thread_pvt *thread;
00214     int last = 0;
00215
00216     if (!(thread = get_thread_from_id())) {
00217         return NULL;
00218     }
00219
00220     for(;;) {
00221         /*if im the last one leave this is done locked to make sure no items are added/removed*/
00222         objlock(threads);
00223         if (!(bucket_list_cnt(threads->list) - last)) {
00224             if (threads->manager) {
00225                 objunref(threads->manager);
00226                 threads->manager = NULL;
00227             }
00228             objunlock(threads);
00229             objunref(thread);
00230             break;
00231         }
00232         objunlock(threads);
00233
00234         /* Ive been joined so i can leave when im alone*/
00235         if (testflag(thread, TL_THREAD_JOIN)) {
00236             clearflag(thread, TL_THREAD_JOIN);
00237             last = 1;
00238         }
00239
00240         /*Cancel all running threads*/
00241         if (testflag(thread, TL_THREAD_STOP)) {
00242             clearflag(thread, TL_THREAD_STOP);
00243             /* Stop any more threads*/
00244             objlock(threads);
00245             if (threads->manager) {
00246                 objunref(threads->manager);
00247                 threads->manager = NULL;
00248             }
00249             objunlock(threads);
00250
00251             /* cancel all threads now that they stoped*/
00252             bucketlist_callback(threads->list, stop_threads, thread);
00253             last = 1;
00254         }
00255 #ifdef __WIN32__
00256         Sleep(1000);
00257 #else
00258         sleep(1);
00259 #endif
00260     }
00261     return NULL;
00262 }
00263
00268 extern int startthreads(void) {
00269     struct threadcontainer *tc;
00270
00271     tc = (objref(threads)) ? threads : NULL;
00272
00273     if (tc) {
00274         objunref(tc);
00275         return 1;
00276     }
00277
00278     if (!(tc = objalloc(sizeof(*threads), close_threads))) {
00279         return 0;
00280     }
00281
00282     if (!tc->list && !(tc->list = create_bucketlist(4, hash_thread))) {
00283         objunref(tc);
00284         return 0;
00285     }
00286
00287     threads = tc;
00288     if (!(tc->manager = framework_mkthread(managethread, manage_clean, manager_sig
      , NULL, THREAD_OPTION_JOINABLE | THREAD_OPTION_RETURN))) {
00289         objunref(tc);
```

```
00290         return 0;
00291     }
00292
00293     return 1;
00294 }
00295
00303 extern void stopthreads(int join) {
00304     struct threadcontainer *tc;
00305
00306     tc = (objref(threads)) ? threads : NULL;
00307     if (!tc) {
00308         return;
00309     }
00310
00311     objlock(tc);
00312     if (tc->manager) {
00313         setflag(tc->manager, TL_THREAD_STOP);
00314         if (join) {
00315             setflag(tc->manager, TL_THREAD_JOIN);
00316             objunlock(tc);
00317             pthread_join(tc->manager->thr, NULL);
00318         } else {
00319             objunlock(tc);
00320         }
00321     } else {
00322         objunlock(tc);
00323     }
00324     objunlock(tc);
00325     objunref(tc);
00326 }
00327
00328 static void thread_cleanup(void *data) {
00329     struct thread_pvt *thread = data;
00330
00331     /*remove from thread list manager unrefs threads in cleanup run 1st*/
00332     remove_bucket_item(threads->list, thread);
00333
00334     /*Run cleanup*/
00335     clearflag(thread, TL_THREAD_RUN);
00336     setflag(thread, TL_THREAD_DONE);
00337     if (thread->cleanup) {
00338         thread->cleanup(thread->data);
00339     }
00340
00341     /*remove thread reference*/
00342     objunref(thread);
00343 }
00344
00345 static void *threadwrap(void *data) {
00346     struct thread_pvt *thread = data;
00347     void *ret = NULL;
00348     int cnt;
00349
00350     objref(thread);
00351
00352     for(cnt = 0;!testflag(thread, TL_THREAD_RUN) && (cnt < 100); cnt++) {
00353         usleep(1000);
00354     }
00355
00356     if (cnt == 100) {
00357         return NULL;
00358     }
00359
00360     pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, NULL);
00361     if (!testflag(thread, TL_THREAD_CAN_CANCEL)) {
00362         pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
00363     }
00364
00365     if (!testflag(thread, TL_THREAD_JOINABLE)) {
00366         pthread_detach(thread->thr);
00367     }
00368
00369     pthread_cleanup_push(thread_cleanup, thread);
00370     ret = thread->func(thread->data);
00371     pthread_cleanup_pop(1);
00372
00373     return (ret);
00374 }
00375
00387 extern struct thread_pvt *framework_mkthread(
      threadfunc func, threadcleanup cleanup,
      threadsighandler sig_handler, void *data, int flags) {
00388     struct thread_pvt *thread;
00389     struct threadcontainer *tc = NULL;
00390
00391     /*Grab a reference for threads in this scope start up if we can*/
00392     if (!(tc = (objref(threads)) ? threads : NULL)) {
```

```
00393         if (!thread_can_start) {
00394             return NULL;
00395         } else if (!startthreads()) {
00396             return NULL;
00397         }
00398         if (!(tc = (objref(threads)) ? threads : NULL)) {
00399             return NULL;
00400         }
00401     }
00402
00403     objlock(tc);
00404     /* dont allow threads if no manager or it not started*/
00405     if ((!tc->manager || !func) && (func != managethread)) {
00406         /*im shuting down*/
00407         objunlock(tc);
00408         objunref(tc);
00409         return NULL;
00410     } else if (!(thread = objalloc(sizeof(*thread), free_thread))) {
00411         /* could not create*/
00412         objunlock(tc);
00413         objunref(tc);
00414         return NULL;
00415     }
00416
00417     thread->data = (objref(data)) ? data : NULL;
00418     thread->flags = flags << 16;
00419     thread->cleanup = cleanup;
00420     thread->sighandler = sig_handler;
00421     thread->func = func;
00422     objunlock(tc);
00423
00424     /* start thread and check it*/
00425     if (pthread_create(&thread->thr, NULL, threadwrap, thread) || pthread_kill(thread->
    thr, 0)) {
00426         objunref(thread);
00427         objunref(tc);
00428         return NULL;
00429     }
00430
00431     /*Activate the thread it needs to be flaged to run or it will die*/
00432     objlock(tc);
00433     addtobucket(tc->list, thread);
00434     setflag(thread, TL_THREAD_RUN);
00435     objunlock(tc);
00436     objunref(tc);
00437
00438     if (testflag(thread, TL_THREAD_RETURN)) {
00439         return thread;
00440     } else {
00441         objunref(thread);
00442         return NULL;
00443     }
00444 }
00445
00450 extern void jointhreads(void) {
00451     struct threadcontainer *tc;
00452
00453     tc = (objref(threads)) ? threads : NULL;
00454     if (!tc) {
00455         return;
00456     }
00457
00458     objlock(tc);
00459     if (tc->manager) {
00460         setflag(tc->manager, TL_THREAD_JOIN);
00461         objunlock(tc);
00462         pthread_join(tc->manager->thr, NULL);
00463     } else {
00464         objunlock(tc);
00465     }
00466     objunref(tc);
00467 }
00468
00469
00470 #ifndef __WIN32
00471 static int handle_thread_signal(struct thread_pvt *thread, int sig) {
00472     int ret;
00473
00474     if (thread->sighandler) {
00475         thread->sighandler(sig, thread->data);
00476         ret = 1;
00477     } else {
00478         ret = -1;
00479     }
00480     return ret;
00481 }
00482 #endif
```

```
00483
00496 extern int thread_signal(int sig) {
00497     int ret = 0;
00498 #ifndef __WIN32
00499     struct thread_pvt *thread = NULL;
00500
00501     if (!(thread = get_thread_from_id())) {
00502         return 0;
00503     }
00504
00505     switch(sig) {
00506         case SIGUSR1:
00507         case SIGUSR2:
00508         case SIGHUP:
00509         case SIGALRM:
00510             ret = handle_thread_signal(thread, sig);
00511             break;
00512         case SIGINT:
00513         case SIGTERM:
00514             ret = handle_thread_signal(thread, sig);
00515     }
00516     objunref(thread);
00517 #endif
00518     return ret;
00519 }
00520
```

14.50 src/unixsock.c File Reference

Attach a thread to a unix socket start a new thread on connect.

```
#include <sys/socket.h>
#include <libgen.h>
#include <sys/stat.h>
#include <linux/un.h>
#include <linux/limits.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "include/dtsapp.h"
```

Data Structures

- struct unixserv_sockthread

  Unix socket server data structure.
- struct unixclient_sockthread

  Unix socket client data structure.

Functions

- struct fwsocket * unixsocket_server (const char *sock, int protocol, int mask, socketrecv read, void *data)

  Create and run UNIX server socket thread.
- struct fwsocket * unixsocket_client (const char *sock, int protocol, socketrecv read, void *data)

  Create a client thread on the socket.

14.50.1 Detailed Description

Attach a thread to a unix socket start a new thread on connect. A thread is started on the sockect and will start a new client thread on each connection with the socket as the data

Definition in file unixsock.c.

14.51 unixsock.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00027 #ifdef __WIN32__
00028 #include <winsock2.h>
00029 #else
00030 #include <sys/socket.h>
00031 #endif
00032 #include <libgen.h>
00033 #include <sys/stat.h>
00034 #include <linux/un.h>
00035 #include <linux/limits.h>
00036 #include <fcntl.h>
00037 #include <errno.h>
00038 #include <unistd.h>
00039 #include <stdio.h>
00040 #include <stdlib.h>
00041 #include <string.h>
00042
00043 #include "include/dtsapp.h"
00044
00046 struct unixserv_sockthread {
00048     struct fwsocket *sock;
00050     char sockpath[UNIX_PATH_MAX+1];
00052     int mask;
00054     int protocol;
00057     socketrecv    read;
00059     void        *data;
00060 };
00061
00063 struct unixclient_sockthread {
00065     struct fwsocket *sock;
00068     socketrecv    client;
00070     const char  *endpoint;
00072     void        *data;
00073 };
00074
00075 /*
00076  * UNIX sock client
00077  */
00078 static void *unsock_client(void *data) {
00079     struct unixclient_sockthread *unsock = data;
00080     struct fwsocket *sock = unsock->sock;
00081     struct  timeval tv;
00082     fd_set  rd_set, act_set;
00083     int selfd;
00084     int on = 1;
00085     int fd, fdf;
00086
00087
00088     FD_ZERO(&rd_set);
00089
00090     fd = sock->sock;
00091     fdf = fcntl(fd, F_GETFL);
00092     fcntl(fd, F_SETFD, fdf | O_NONBLOCK);
00093     /*enable passing credentials*/
00094     setsockopt(fd, SOL_SOCKET, SO_PASSCRED, &on, sizeof(on));
00095     FD_SET(fd, &rd_set);
00096
00097     while (framework_threadok()) {
00098         act_set = rd_set;
00099         tv.tv_sec = 0;
00100         tv.tv_usec = 20000;
00101
```

```
00102         selfd = select(fd + 1, &act_set, NULL, NULL, &tv);
00103
00104         /*returned due to interupt continue or timed out*/
00105         if ((selfd < 0 && errno == EINTR) || (!selfd)) {
00106             continue;
00107         } else if (selfd < 0) {
00108             break;
00109         }
00110
00111         if (FD_ISSET(sock->sock, &act_set) && unsock->client) {
00112             unsock->client(sock, unsock->data);
00113         }
00114     }
00115     objunref(unsock);
00116
00117     return NULL;
00118 }
00119
00120 static void unixclient_sockthread_free(void *data) {
00121     struct unixclient_sockthread *uc = data;
00122
00123     if (uc->sock) {
00124         objunref(uc->sock);
00125     }
00126     if (uc->data) {
00127         objunref(uc->data);
00128     }
00129     if (uc->endpoint) {
00130         if (!strlenzero(uc->endpoint)) {
00131             unlink(uc->endpoint);
00132         }
00133         free((void*)uc->endpoint);
00134     }
00135 }
00136
00137 static int new_unixclientthread(struct fwsocket *fws, const char *
      endpoint, socketrecv read, void *data) {
00138     struct unixclient_sockthread *unsock;
00139     void *thread;
00140
00141     if (!(unsock = objalloc(sizeof(*unsock), unixclient_sockthread_free))) {
00142         return 0;
00143     }
00144
00145     unsock->sock = fws;
00146     unsock->data = (objref(data)) ? data : NULL;
00147     unsock->client = read;
00148     unsock->endpoint = endpoint;
00149
00150     if (!(thread = framework_mkthread(unsock_client, NULL, NULL, unsock,
      THREAD_OPTION_RETURN))) {
00151         objunref(unsock);
00152         return 0;
00153     }
00154     objunref(thread);
00155     return 1;
00156 }
00157
00158 /*
00159  * UNIX sock server
00160  */
00161 static void *unsock_serv(void *data) {
00162     struct unixserv_sockthread *unsock = data;
00163     struct fwsocket *newsock, *sock;
00164     union sockstruct *adr;
00165     unsigned int salen;
00166     struct  timeval tv;
00167     fd_set  rd_set, act_set;
00168     int selfd;
00169     int on = 1;
00170     int fd, fdf;
00171
00172     /* set user RW */
00173     umask(unsock->mask);
00174
00175
00176     sock = unsock->sock;
00177     sock->flags |= SOCK_FLAG_UNIX;
00178     fd = sock->sock;
00179
00180     fdf = fcntl(fd, F_GETFL);
00181     fcntl(fd, F_SETFD, fdf | O_NONBLOCK);
00182
00183     adr = &sock->addr;
00184     memset(&adr->un, 0, sizeof(adr->un));
00185     adr->un.sun_family = PF_UNIX;
00186     salen = sizeof(adr->un);
```

```
00187     strncpy((char *)adr->un.sun_path, unsock->sockpath, sizeof(adr->un.sun_path) -1);
00188
00189     /*enable passing credentials*/
00190     setsockopt(fd, SOL_SOCKET, SO_PASSCRED, &on, sizeof(on));
00191
00192     if (bind(fd, (struct sockaddr *)&adr->un, salen)) {
00193         if (errno == EADDRINUSE) {
00194             /* delete old file*/
00195             unlink(unsock->sockpath);
00196             if (bind(fd, (struct sockaddr *)&adr->un, sizeof(struct sockaddr_un))) {
00197                 objunref(unsock);
00198                 close(fd);
00199                 return NULL;
00200             }
00201         } else {
00202             objunref(unsock);
00203             close(fd);
00204             return NULL;
00205         }
00206     }
00207
00208     if (unsock->protocol == SOCK_STREAM) {
00209         if (listen(fd, 10)) {
00210             close(fd);
00211             objunref(unsock);
00212             return NULL;
00213         }
00214     }
00215
00216     FD_ZERO(&rd_set);
00217     FD_SET(fd, &rd_set);
00218
00219     while (framework_threadok()) {
00220         act_set = rd_set;
00221         tv.tv_sec = 0;
00222         tv.tv_usec = 20000;
00223
00224         selfd = select(fd + 1, &act_set, NULL, NULL, &tv);
00225
00226         /*returned due to interupt continue or timed out*/
00227         if ((selfd < 0 && errno == EINTR) || (!selfd)) {
00228             continue;
00229         } else if (selfd < 0) {
00230             break;
00231         }
00232
00233         if (FD_ISSET(fd, &act_set)) {
00234             if (unsock->protocol == SOCK_STREAM) {
00235                 if ((newsock = accept_socket(sock))) {
00236                     if (!(new_unixclientthread(newsock, NULL, unsock->read, unsock->
      data))) {
00237                         objunref(newsock);
00238                     }
00239                 }
00240             } else if (unsock->read) {
00241                 unsock->read(sock, unsock->data);
00242
00243             }
00244         }
00245     }
00246
00247     close(fd);
00248     objunref(unsock);
00249
00250     return NULL;
00251 }
00252
00253 static void free_unixserv(void *data) {
00254     struct unixserv_sockthread *unsock = data;
00255
00256     if (unsock->sock) {
00257         objunref(unsock->sock);
00258     }
00259
00260     if (!strlenzero(unsock->sockpath)) {
00261         unlink(unsock->sockpath);
00262     }
00263
00264     if (unsock->data) {
00265         objunref(data);
00266     }
00267 }
00268
00277 extern struct fwsocket *unixsocket_server(const char *sock, int protocol, int mask
      , socketrecv read, void *data) {
00278     struct unixserv_sockthread *unsock;
00279
```

```
00280     if (!(unsock = objalloc(sizeof(*unsock), free_unixserv))) {
00281         return NULL;
00282     }
00283
00284     strncpy(unsock->sockpath, sock, UNIX_PATH_MAX);
00285     unsock->mask = mask;
00286     unsock->read = read;
00287     unsock->protocol = protocol;
00288     unsock->data = (objref(data)) ? data : NULL;
00289
00290     /*Create a UNIX socket structure*/
00291     if (!(unsock->sock = make_socket(PF_UNIX, protocol, 0, NULL))) {
00292         objunref(unsock);
00293         return NULL;
00294     }
00295
00296     framework_mkthread(unsock_serv, NULL, NULL, unsock, 0);
00297     return (objref(unsock->sock)) ? unsock->sock : NULL;
00298 }
00299
00310 extern struct fwsocket *unixsocket_client(const char *sock, int protocol,
    socketrecv read, void *data) {
00311     struct fwsocket *fws;
00312     union sockstruct caddr, *saddr;
00313     char *temp = NULL;
00314     const char *tmpsock;
00315     int salen;
00316     mode_t omask;
00317
00318     /*Create a UNIX socket structure*/
00319     if (!(fws = make_socket(PF_UNIX, protocol, 0, NULL))) {
00320         return NULL;
00321     }
00322
00323     /* bind my endpoint to temp file*/
00324     if (protocol == SOCK_DGRAM) {
00325         /*yip i want only a inode here folks*/
00326         omask = umask(S_IXUSR | S_IRUSR | S_IWUSR | S_IWGRP | S_IRGRP | S_IXGRP | S_IWOTH | S_IROTH |
    S_IXOTH);
00327         tmpsock = basename((char*)sock);
00328         temp = tempnam(NULL, tmpsock);
00329         if (strlenzero(temp)) {
00330             if (temp) {
00331                 free(temp);
00332             }
00333             objunref(fws);
00334             return NULL;
00335         }
00336
00337         /*Allocate address and connect to the client*/
00338         salen = sizeof(caddr.un);
00339         memset(&caddr.un, 0, salen);
00340         caddr.un.sun_family = PF_UNIX;
00341         strncpy((char *)caddr.un.sun_path, temp, sizeof(caddr.un.sun_path) -1);
00342
00343         if (bind(fws->sock, (struct sockaddr *)&caddr.un, salen)) {
00344             /*reset umask*/
00345             umask(omask);
00346             if (temp) {
00347                 if (!strlenzero(temp)) {
00348                     unlink(temp);
00349                 }
00350                 free(temp);
00351             }
00352             objunref(fws);
00353             return NULL;
00354         }
00355         /*reset umask*/
00356         umask(omask);
00357     }
00358
00359     /*Allocate address and connect to the server*/
00360     saddr =  &fws->addr;
00361     salen = sizeof(saddr->un);
00362     memset(&saddr->un, 0, salen);
00363     saddr->un.sun_family = PF_UNIX;
00364     strncpy((char *)saddr->un.sun_path, sock, sizeof(saddr->un.sun_path) -1);
00365
00366     if (connect(fws->sock, (struct sockaddr *)&saddr->un, salen)) {
00367         if (temp) {
00368             if (!strlenzero(temp)) {
00369                 unlink(temp);
00370             }
00371             free(temp);
00372         }
00373         objunref(fws);
00374         return NULL;
```

```
00375     }
00376
00377     fws->flags |= SOCK_FLAG_UNIX;
00378     if (!(new_unixclientthread(fws, temp, read, data))) {
00379         if (temp) {
00380             if (!strlenzero(temp)) {
00381                 unlink(temp);
00382             }
00383             free(temp);
00384         }
00385         objunref(fws);
00386         return NULL;
00387     }
00388
00389     return (objref(fws)) ? fws : NULL;
00390 }
00391
```

14.52 src/util.c File Reference

Utilities commonly used.

```
#include <openssl/bio.h>
#include <openssl/buffer.h>
#include <openssl/evp.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <openssl/rand.h>
#include <openssl/md5.h>
#include <openssl/sha.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <sys/time.h>
#include "include/dtsapp.h"
```

Functions

- void seedrand (void)

  Seed openssl random number generator.
- int genrand (void *buf, int len)

  Generate random sequence.
- void sha512sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA2-512 hash accross 2 data chunks.
- void sha512sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA2-512 hash.
- void sha256sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA2-256 hash accross 2 data chunks.
- void sha256sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA2-256 hash.
- void sha1sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the SHA1 hash accross 2 data chunks.
- void sha1sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the SHA1 hash.

- void md5sum2 (unsigned char *buff, const void *data, unsigned long len, const void *data2, unsigned long len2)

  Calculate the MD5 hash accross 2 data chunks.
- void md5sum (unsigned char *buff, const void *data, unsigned long len)

  Calculate the MD5 hash.
- int md5cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two md5 hashes.
- int sha1cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA1 hashes.
- int sha256cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA2-256 hashes.
- int sha512cmp (unsigned char *digest1, unsigned char *digest2)

  Compare two SHA2-512 hashes.
- void md5hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) MD5.
- void sha1hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA1.
- void sha256hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA2-256.
- void sha512hmac (unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long klen)

  Hash Message Authentication Codes (HMAC) SHA2-512.
- int strlenzero (const char *str)

  Check if a string is zero length.
- char * ltrim (char *str)

  Trim white space at the begining of a string.
- char * rtrim (const char *str)

  Trim white space at the end of a string.
- char * trim (const char *str)

  Trim whitesapce from the beggining and end of a string.
- uint64_t tvtontp64 (struct timeval *tv)

  Convert a timeval struct to 64bit NTP time.
- uint16_t checksum (const void *data, int len)

  Obtain the checksum for a buffer.
- uint16_t checksum_add (const uint16_t checksum, const void *data, int len)

  Obtain the checksum for a buffer adding a checksum.
- uint16_t verifysum (const void *data, int len, const uint16_t check)

  Verify a checksum.
- void touch (const char *filename, uid_t user, gid_t group)

  Create a file and set user and group.
- char * b64enc_buf (const char *message, uint32_t len, int nonl)

  Base 64 encode a buffer.
- char * b64enc (const char *message, int nonl)

  Base 64 encode a string.

14.52.1 Detailed Description

Utilities commonly used.

Definition in file util.c.

14.53 util.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00019 #include <openssl/bio.h>
00020 #include <openssl/buffer.h>
00021 #include <openssl/evp.h>
00022
00046 #ifdef __WIN32__
00047 #include <winsock2.h>
00048 #include <windows.h>
00049 #endif
00050
00051 #include <fcntl.h>
00052 #include <string.h>
00053 #include <unistd.h>
00054 #include <openssl/rand.h>
00055 #include <openssl/md5.h>
00056 #include <openssl/sha.h>
00057 #include <ctype.h>
00058 #include <stdint.h>
00059 #include <stdio.h>
00060 #include <sys/time.h>
00061
00062 #include "include/dtsapp.h"
00063
00068 extern void seedrand(void) {
00069     int fd = open("/dev/random", O_RDONLY);
00070     int len;
00071     char    buf[64];
00072
00073     len = read(fd, buf, 64);
00074     RAND_seed(buf, len);
00075 }
00076
00082 extern int genrand(void *buf, int len) {
00083     return (RAND_bytes(buf, len));
00084 }
00085
00097 extern void sha512sum2(unsigned char *buff, const void *data, unsigned long len, const void *
      data2, unsigned long len2) {
00098     SHA512_CTX c;
00099
00100     SHA512_Init(&c);
00101     SHA512_Update(&c, data, len);
00102     if (data2) {
00103         SHA512_Update(&c, data2, len2);
00104     }
00105     SHA512_Final(buff, &c);
00106 }
00107
00114 extern void sha512sum(unsigned char *buff, const void *data, unsigned long len) {
00115     sha512sum2(buff, data, len, NULL, 0);
00116 }
00117
00118
00127 extern void sha256sum2(unsigned char *buff, const void *data, unsigned long len, const void *
      data2, unsigned long len2) {
00128     SHA256_CTX c;
00129
00130     SHA256_Init(&c);
00131     SHA256_Update(&c, data, len);
00132     if (data2) {
00133         SHA256_Update(&c, data2, len2);
00134     }
00135     SHA256_Final(buff, &c);
00136 }
00137
00144 extern void sha256sum(unsigned char *buff, const void *data, unsigned long len) {
00145     sha256sum2(buff, data, len, NULL, 0);
```

```
00146 }
00147
00156 extern void sha1sum2(unsigned char *buff, const void *data, unsigned long len, const void *data2,
      unsigned long len2) {
00157     SHA_CTX c;
00158
00159     SHA_Init(&c);
00160     SHA_Update(&c, data, len);
00161     if (data2) {
00162         SHA_Update(&c, data2, len2);
00163     }
00164     SHA_Final(buff, &c);
00165 }
00166
00173 extern void sha1sum(unsigned char *buff, const void *data, unsigned long len) {
00174     sha1sum2(buff, data, len, NULL, 0);
00175 }
00176
00185 extern void md5sum2(unsigned char *buff, const void *data, unsigned long len, const void *data2,
      unsigned long len2) {
00186     MD5_CTX c;
00187
00188     MD5_Init(&c);
00189     MD5_Update(&c, data, len);
00190     if (data2) {
00191         MD5_Update(&c, data2, len2);
00192     }
00193     MD5_Final(buff, &c);
00194 }
00195
00202 extern void md5sum(unsigned char *buff, const void *data, unsigned long len) {
00203     md5sum2(buff, data, len, NULL, 0);
00204 }
00205
00206 static int _digest_cmp(unsigned char *md51, unsigned char *md52, int len) {
00207     int cnt;
00208     int chk = 0;
00209
00210     for(cnt = 0; cnt < len; cnt ++) {
00211         chk += md51[cnt] & ~md52[cnt];
00212     }
00213
00214     return (chk);
00215 }
00216
00223 extern int md5cmp(unsigned char *digest1, unsigned char *digest2) {
00224     return (_digest_cmp(digest1, digest2, 16));
00225 }
00226
00233 extern int sha1cmp(unsigned char *digest1, unsigned char *digest2) {
00234     return (_digest_cmp(digest1, digest2, 20));
00235 }
00236
00243 extern int sha256cmp(unsigned char *digest1, unsigned char *digest2) {
00244     return (_digest_cmp(digest1, digest2, 32));
00245 }
00246
00253 extern int sha512cmp(unsigned char *digest1, unsigned char *digest2) {
00254     return (_digest_cmp(digest1, digest2, 64));
00255 }
00256
00257 static void _hmac(unsigned char *buff, const void *data, unsigned long len, const void *key, unsigned long
      klen,
00258             void (*func)(unsigned char *, const void *, unsigned long, const void *, unsigned long), short
      alglen) {
00259     unsigned char   okey[64], ikey[64];
00260     int     bcnt;
00261
00262     memset(ikey, 0, 64);
00263     memset(okey, 0, 64);
00264
00265     if (klen < 64) {
00266         memcpy(ikey, key, klen);
00267         memcpy(okey, key, klen);
00268     } else {
00269         md5sum(okey, key, klen);
00270         memcpy(ikey, okey, klen);
00271     }
00272
00273     for (bcnt = 0; bcnt < 64; bcnt++) {
00274         ikey[bcnt] ^= 0x36;
00275         okey[bcnt] ^= 0x5c;
00276     };
00277
00278     func(buff, ikey, 64, data, len);
00279     func(buff, okey, 64, buff, alglen);
00280 }
```

```
00281
00290 extern void md5hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
      unsigned long klen) {
00291     _hmac(buff, data, len, key, klen, md5sum2, 16);
00292 }
00293
00302 extern void sha1hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
      unsigned long klen) {
00303     _hmac(buff, data, len, key, klen, sha1sum2, 20);
00304 }
00305
00314 extern void sha256hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
       unsigned long klen) {
00315     _hmac(buff, data, len, key, klen, sha256sum2, 32);
00316 }
00317
00326 extern void sha512hmac(unsigned char *buff, const void *data, unsigned long len, const void *key,
       unsigned long klen) {
00327     _hmac(buff, data, len, key, klen, sha512sum2, 64);
00328 }
00329
00341 extern int strlenzero(const char *str) {
00342     if (str && strlen(str)) {
00343         return (0);
00344     }
00345     return (1);
00346 }
00347
00348
00353 extern char *ltrim(char *str) {
00354     char *cur = str;
00355
00356     if (strlenzero(str)) {
00357         return (str);
00358     }
00359
00360     while(isspace(cur[0])) {
00361         cur++;
00362     }
00363
00364     return (cur);
00365 }
00366
00367
00372 extern char *rtrim(const char *str) {
00373     int len;
00374     char *cur = (char *)str;
00375
00376     if (strlenzero(str)) {
00377         return (cur);
00378     }
00379
00380     len = strlen(str) - 1;
00381     while(len && isspace(cur[len])) {
00382         cur[len] = '\0';
00383         len--;
00384     }
00385
00386     return (cur);
00387 }
00388
00393 extern char *trim(const char *str) {
00394     char *cur = (char *)str;
00395
00396     cur = ltrim(cur);
00397     cur = rtrim(cur);
00398     return (cur);
00399 }
00400
00405 extern uint64_t tvtontp64(struct timeval *tv) {
00406     return ((((uint64_t)tv->tv_sec + 2208988800u) << 32) + ((uint32_t)tv->tv_usec * 4294.967296));
00407 }
00408
00409 /*
00410  * RFC 1701 Checksum based on code from the RFC
00411  */
00412 static uint16_t _checksum(const void *data, int len, const uint16_t check) {
00413     uint64_t csum = 0;
00414     const uint32_t *arr = (uint32_t *)data;
00415
00416     /*handle 32bit chunks*/
00417     while(len > 3) {
00418         csum += *arr++;
00419         len -= 4;
00420     }
00421
00422     /*handle left over 16 bit chunk*/
```

```
00423     if (len > 1) {
00424         csum += *(uint16_t *)arr;
00425         arr = (uint32_t *)((uint16_t *)arr + 1);
00426         len -= 2;
00427     }
00428 
00429     /*handle odd byte*/
00430     if (len) {
00431         csum += *(uint8_t *)arr;
00432     }
00433 
00434     /*add checksum when called as verify*/
00435     if (check) {
00436         csum += check;
00437     }
00438 
00439     /*collapse to 16 bits adding all overflows leaving 16bit checksum*/
00440     while(csum >> 16) {
00441         csum = (csum & 0xffff) + (csum >> 16);
00442     }
00443 
00444     return (~(uint16_t)csum);
00445 }
00446 
00452 extern uint16_t checksum(const void *data, int len) {
00453     return (_checksum(data, len, 0));
00454 }
00455 
00456 
00463 extern uint16_t checksum_add(const uint16_t checksum, const void *data, int len) {
00464     return (_checksum(data, len, ~checksum));
00465 }
00466 
00473 extern uint16_t verifysum(const void *data, int len, const uint16_t check) {
00474     return (_checksum(data, len, check));
00475 }
00476 
00483 #ifndef __WIN32__
00484 extern void touch(const char *filename, uid_t user, gid_t group) {
00485     int res;
00486 #else
00487 extern void touch(const char *filename) {
00488 #endif
00489     int fd;
00490 
00491     fd = creat(filename, 0600);
00492     close(fd);
00493 #ifndef __WIN32__
00494     res = chown(filename, user, group);
00495     res++;
00496 #endif
00497     return;
00498 }
00499 
00506 extern char *b64enc_buf(const char *message, uint32_t len, int nonl) {
00507     BIO *bmem, *b64;
00508     BUF_MEM *ptr;
00509     char *buffer;
00510     double encodedSize;
00511 
00512     encodedSize = 1.36*len;
00513     buffer = objalloc(encodedSize+1, NULL);
00514 
00515     b64 = BIO_new(BIO_f_base64());
00516     bmem = BIO_new(BIO_s_mem());
00517     b64 = BIO_push(b64, bmem);
00518     if (nonl) {
00519         BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL);
00520     }
00521     BIO_write(b64, message, len);
00522     BIO_flush(b64);
00523     BIO_get_mem_ptr(b64, &ptr);
00524 
00525     buffer = objalloc(ptr->length+1, NULL);
00526     memcpy(buffer, ptr->data, ptr->length);
00527 
00528 
00529     BIO_free_all(b64);
00530 
00531     return buffer;
00532 }
00533 
00539 extern char *b64enc(const char *message, int nonl) {
00540     return b64enc_buf(message, strlen(message), nonl);
00541 }
00542 
```

14.54 src/winiface.cpp File Reference

Various routines for supporting Windows also requires C++.

```
#include <stdio.h>
#include <stdint.h>
#include "include/dtsapp.h"
```

Functions

- const char * inet_ntop (int af, const void *src, char *dest, socklen_t size)

  Win32 implementation of inet_ntop.
- struct ifinfo * get_ifinfo (const char *iface)

  Return interface info for a specified interface.

14.54.1 Detailed Description

Various routines for supporting Windows also requires C++.

Definition in file winiface.cpp.

14.55 winiface.cpp

```
00001 #include <stdio.h>
00002 #include <stdint.h>
00003 #include "include/dtsapp.h"
00004
00011 static PIP_ADAPTER_ADDRESSES get_adaptorinfo(int obufsize, int tries) {
00012     PIP_ADAPTER_ADDRESSES ainfo = NULL;
00013     int i = 1;
00014     unsigned long buflen;
00015
00016     buflen = obufsize * i;
00017
00018     do {
00019         if (!(ainfo = (IP_ADAPTER_ADDRESSES *)malloc(buflen))) {
00020             return NULL;
00021         }
00022
00023             if (GetAdaptersAddresses(AF_UNSPEC, GAA_FLAG_INCLUDE_PREFIX, NULL, ainfo, &buflen) ==
      ERROR_BUFFER_OVERFLOW) {
00024             free(ainfo);
00025             ainfo = NULL;
00026         } else {
00027             break;
00028         }
00029
00030         i++;
00031     } while (i <= tries);
00032
00033     return ainfo;
00034 }
00035
00043 const char *inet_ntop(int af, const void *src, char *dest, socklen_t size) {
00044     union sockstruct sa;
00045     int res = 0;
00046     char serv[NI_MAXSERV];
00047
00048     memset(&sa, 0, sizeof(sa));
00049     sa.ss.ss_family = af;
00050
00051     switch(af) {
00052         case AF_INET:
00053             memcpy(&sa.sa4.sin_addr, src, sizeof(struct in_addr));
00054             res = getnameinfo(&sa.sa, sizeof(struct sockaddr_in), dest, size, serv, NI_MAXSERV,
      NI_NUMERICHOST | NI_NUMERICSERV);
00055             break;
00056         case AF_INET6:
00057             memcpy(&sa.sa6.sin6_addr, src, sizeof(struct in6_addr));
00058             res = getnameinfo(&sa.sa, sizeof(struct sockaddr_in6), dest, size, serv, NI_MAXSERV,
```

```
    NI_NUMERICHOST | NI_NUMERICSERV);
00059             break;
00060     }
00061     return (!res) ? dest : NULL;
00062 }
00063
00064 static void free_ifinfo(void *data) {
00065     struct ifinfo *ifinf = (struct ifinfo*)data;
00066
00067     if (ifinf->ifaddr) {
00068         free((void*)ifinf->ifaddr);
00069     }
00070     if (ifinf->ipv4addr) {
00071         free((void*)ifinf->ipv4addr);
00072     }
00073     if (ifinf->ipv6addr) {
00074         free((void*)ifinf->ipv6addr);
00075     }
00076 }
00077
00078
00083 struct ifinfo *get_ifinfo(const char *iface) {
00084     PIP_ADAPTER_ADDRESSES ainfo = NULL, cinfo;
00085     PIP_ADAPTER_UNICAST_ADDRESS pUnicast;
00086     struct sockaddr_storage *ss;
00087     char tmphn[NI_MAXHOST];
00088     char host4[NI_MAXHOST];
00089     char host6[NI_MAXHOST];
00090     int score4 = 0, score6 = 0, nscore;
00091     struct ifinfo *ifinf = NULL;
00092
00093     if (!(ainfo = get_adaptorinfo(15000, 3))) {
00094         return NULL;
00095     }
00096
00097     for(cinfo = ainfo; cinfo; cinfo = cinfo->Next) {
00098         if (strcmp(cinfo->AdapterName, iface)) {
00099             continue;
00100         }
00101
00102         if (!(ifinf = (struct ifinfo*)objalloc(sizeof(*ifinf),free_ifinfo))) {
00103             return NULL;
00104         }
00105
00106         ifinf->idx = (int)cinfo->IfIndex;
00107
00108         if (cinfo->PhysicalAddressLength == 6) {
00109             unsigned int i;
00110             char tmp[4];
00111             char tmp2[18] = "";
00112             for (i = 0; i < cinfo->PhysicalAddressLength; i++) {
00113                 if (i == (cinfo->PhysicalAddressLength - 1)) {
00114                     sprintf(tmp,"%.2X", (int)cinfo->PhysicalAddress[i]);
00115                 } else {
00116                     sprintf(tmp,"%.2X:", (int)cinfo->PhysicalAddress[i]);
00117                         }
00118                 strcat(tmp2, tmp);
00119             }
00120             ifinf->ifaddr = strdup(tmp2);
00121         } else {
00122             ifinf->ifaddr = NULL;
00123         }
00124
00125         for (pUnicast = cinfo->FirstUnicastAddress; pUnicast ;pUnicast = pUnicast->Next) {
00126             ss = (struct sockaddr_storage*)pUnicast->Address.lpSockaddr;
00127             switch(ss->ss_family) {
00128                 case AF_INET:
00129                     nscore = score_ipv4((struct sockaddr_in*)ss, tmphn, NI_MAXHOST);
00130                     if (score4 < nscore) {
00131                         score4 = nscore;
00132                         strcpy(host4, tmphn);
00133                     }
00134                     break;
00135                 case AF_INET6:
00136                     nscore = score_ipv6((struct sockaddr_in6*)ss, tmphn, NI_MAXHOST);
00137                     if (score6 < nscore) {
00138                         score6 = nscore;
00139                         strcpy(host6, tmphn);
00140                     }
00141                     break;
00142             }
00143         }
00144         ifinf->ipv4addr = (strlenzero(host4)) ? NULL : strdup(host4);
00145         ifinf->ipv6addr = (strlenzero(host6)) ? NULL : strdup(host6);
00146         break;
00147     }
00148
```

```
00149     if (ainfo) {
00150         free(ainfo);
00151     }
00152
00153     return ifinf;
00154 }
00155
00156
```

14.56 src/zlib.c File Reference

Simplified implementation of zlib functions.

```
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <zlib.h>
#include "include/dtsapp.h"
```

Functions

- struct zobj * zcompress (uint8_t *buff, uint16_t len, uint8_t level)

  Allocate a buffer and return it with compressed data.
- void zuncompress (struct zobj *buff, uint8_t *obuff)

  Uncompress zobj buffer to buffer.
- int is_gzip (uint8_t *buf, int buf_size)

  check a buffer if it contains gzip magic
- uint8_t * gzinflatebuf (uint8_t *buf_in, int buf_size, uint32_t *len)

  Ungzip a buffer.

14.56.1 Detailed Description

Simplified implementation of zlib functions.

Definition in file zlib.c.

14.57 zlib.c

```
00001 /*
00002 Copyright (C) 2012  Gregory Nietsky <gregory@distrotetch.co.za>
00003         http://www.distrotech.co.za
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation, either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program.  If not, see <http://www.gnu.org/licenses/>.
00017 */
00018
00024 #include <stdint.h>
00025 #include <stdlib.h>
00026 #include <string.h>
00027 #include <zlib.h>
00028
00029 #include "include/dtsapp.h"
00030
```

```
00031 static const unsigned char gzipMagicBytes[] = { 0x1f, 0x8b, 0x08, 0x00 };
00032
00033 static void zobj_free(void *data) {
00034     struct zobj *zdata = data;
00035
00036     if (zdata->buff) {
00037         free(zdata->buff);
00038     }
00039 }
00040
00047 extern struct zobj *zcompress(uint8_t *buff, uint16_t len, uint8_t level) {
00048     struct zobj *ret;
00049
00050     if (!(ret = objalloc(sizeof(*ret), zobj_free))) {
00051         return (NULL);
00052     }
00053
00054     ret->zlen = compressBound(len);
00055     ret->olen = len;
00056
00057     if (!(ret->buff = malloc(ret->zlen))) {
00058         return (NULL);
00059     }
00060     compress2(ret->buff, (uLongf *)&ret->zlen, buff, len, level);
00061
00062     return (ret);
00063 }
00064
00071 extern void zuncompress(struct zobj *buff, uint8_t *obuff) {
00072     uLongf olen = buff->olen;
00073
00074     if (!obuff) {
00075         return;
00076     }
00077
00078     uncompress(obuff, &olen, buff->buff, buff->zlen);
00079 }
00080
00085 extern int is_gzip(uint8_t *buf, int buf_size) {
00086     if (buf_size < 4) {
00087         return 0;
00088     }
00089     if (memcmp(buf, gzipMagicBytes, 4)) {
00090         return 0;
00091     }
00092     return 1;
00093 }
00094
00101 extern uint8_t *gzinflatebuf(uint8_t *buf_in, int buf_size, uint32_t *len) {
00102     z_stream zdat;
00103     uint8_t *buf = NULL, *tmp;
00104     int res;
00105
00106     zdat.opaque = NULL;
00107     zdat.zalloc = NULL;
00108     zdat.zfree = NULL;
00109
00110     zdat.next_in = buf_in;
00111     zdat.avail_in = buf_size;
00112     zdat.next_out = buf;
00113     zdat.avail_out = 0;
00114     zdat.total_out = 0;
00115
00116     if (inflateInit2(&zdat, 31)) {
00117         return NULL;
00118     }
00119
00120     do {
00121         if (!(tmp = realloc(buf,zdat.total_out + (zdat.avail_in * 5) + 1))) {
00122             res = Z_MEM_ERROR;
00123             break;
00124         } else {
00125             buf = tmp;
00126         }
00127         buf[zdat.total_out] = '\0';
00128         zdat.next_out = &buf[zdat.total_out];
00129         zdat.avail_out += zdat.avail_in * 5;
00130     } while ((res = inflate(&zdat, Z_NO_FLUSH)) == Z_OK);
00131
00132     if (res == Z_STREAM_END) {
00133         buf = realloc(buf, zdat.total_out);
00134         *len = zdat.total_out;
00135     } else {
00136         free(buf);
00137         *len = 0;
00138         buf = NULL;
00139     }
```

```
00140     inflateEnd(&zdat);
00141
00142     return buf;
00143 }
00144
```

Chapter 15

Example Documentation

15.1 socket.c

```
#ifdef __WIN32
#include <winsock2.h>
#include <stdint.h>
#else
#include <fcntl.h>
#endif

#include <string.h>
#include <stdio.h>

#include <openssl/ssl.h>
#include <dtsapp.h>

void accept_func(struct fwsocket *sock, void *data) {
}

void server_func(struct fwsocket *sock, void *data) {
    char buff[128];
    union sockstruct addr;

    if (socketread_d(sock, &buff, 128, &addr) > 0) {
        socketwrite_d(sock, &buff, strlen(buff) + 1, &addr);
        printf("[S] %s %i\n", buff, sock->sock);
        sleep(1);
    }
}

void client_func(struct fwsocket *sock, void *data) {
    char buff[128];

    if (socketread(sock, &buff, 128) > 0) {
        socketwrite(sock, &buff, strlen(buff) + 1);
        printf("[C] %s %i\n", buff, sock->sock);
    }
}

void socktest(const char *ipaddr, int tcp, int ssl) {
    struct fwsocket *serv, *client, *client2;
    void *ssl_c = NULL, *ssl_s = NULL, *ssl_c2 = NULL;
    char *buff = "client 1";
    char *buff2 = "client 2";
    int cnt;

    if (ssl && tcp) {
        ssl_s = sslv3_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
      SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
        ssl_c = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
        ssl_c2 = sslv3_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
      SSL_VERIFY_NONE);
    } else if (ssl) {
        ssl_s = dtlsv1_init("certs/cacert.pem", "certs/server-cert.pem", "certs/server-key.pem",
     SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
        ssl_c = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem",
     SSL_VERIFY_NONE);
        ssl_c2 = dtlsv1_init("certs/cacert.pem", "certs/client-cert.pem", "certs/client-key.pem"
      , SSL_VERIFY_NONE);
    }
```

```
    if (tcp) {
        serv = tcpbind(ipaddr, "1111", ssl_s, 10);
        client = tcpconnect(ipaddr, "1111", ssl_c);
        client2 = tcpconnect(ipaddr, "1111", ssl_c2);
    } else {
        serv = udpbind(ipaddr, "1111", ssl_s);
        client = udpconnect(ipaddr, "1111", ssl_c);
        client2 = udpconnect(ipaddr, "1111", ssl_c2);
    }

    if (serv && client && client2) {
        socketserver(serv, server_func, accept_func, NULL, NULL);
        socketclient(client, NULL, client_func, NULL);
        socketclient(client2, NULL, client_func, NULL);

        socketwrite(client, buff, strlen(buff)+1);
        socketwrite(client2, buff2, strlen(buff2)+1);

        sleep(5);
    } else {
        printf("ERROR\n");
    }

    close_socket(client);
    close_socket(client2);
    close_socket(serv);
}

#ifndef __WIN32
void unixsocktest(const char *socket, int protocol) {
    char *buff = "client 1";
    char *buff2 = "client 2";
    struct fwsocket *client, *client2, *server;

    server = unixsocket_server(socket, protocol, S_IXUSR | S_IWGRP | S_IRGRP | S_IXGRP |
      S_IWOTH | S_IROTH | S_IXOTH, server_func, NULL);
    sleep(1); /*wait for socket*/
    client = unixsocket_client(socket, protocol, client_func, NULL);
    client2 = unixsocket_client(socket, protocol, client_func, NULL);

    socketwrite_d(client, buff, strlen(buff)+1, NULL);
    socketwrite_d(client2, buff2, strlen(buff2)+1, NULL);

    sleep(5);

    close_socket(client);
    close_socket(client2);
    close_socket(server);
}
#endif

FRAMEWORK_MAIN("Socket Client/Server Echo (TCP/TLS/UDP/DTLS)", "Gregory Hinton Nietsky", "
      gregory@distrotech.co.za",
        "http://www.distrotech.co.za", 2013, "/var/run/sockettest",
      FRAMEWORK_FLAG_DAEMONLOCK, NULL) {

    if (argc < 3) {
#ifndef __WIN32
        printf("Requires arguments %s [tcp|tls|udp|dtls|unix_d|unix_s] [ipaddr|socket]\n", argv[0]);
#else
        printf("Requires arguments %s [tcp|tls|udp|dtls] ipaddr\n", argv[0]);
#endif
        return (-1);
    }

    daemonize();
    if (!strcmp(argv[1], "udp")) {
        socktest(argv[2], 0, 0);
    } else if (!strcmp(argv[1], "dtls")) {
        socktest(argv[2], 0, 1);
    } else if (!strcmp(argv[1], "tcp")) {
        socktest(argv[2], 1, 0);
    } else if (!strcmp(argv[1], "tls")) {
        socktest(argv[2], 1, 1);
#ifndef __WIN32
    } else if (!strcmp(argv[1], "unix_d")) {
        unixsocktest(argv[2], SOCK_DGRAM);
    } else if (!strcmp(argv[1], "unix_s")) {
        unixsocktest(argv[2], SOCK_STREAM);
#endif
    } else {
        printf("Invalid Option\n");
    }
}
```

Index

ALLOC_CONST
  Distrotech Application Library, 29
accept_func
  doxygen/examples/socket.c, 310
accept_socket
  Network socket interface, 65
acctport
  radius_server, 287
add_radserver
  Radius client interface, 119
addr
  fwsocket, 265
addradattrint
  Radius client interface, 119
addradattrip
  Radius client interface, 119
addradattrstr
  Radius client interface, 120
addtobucket
  Hashed bucket linked lists of referenced objects, 47
adji
  natmap, 281
adjo
  natmap, 281
attr
  ldap_modreq, 276
attrs
  ldap_entry, 274
  radius_packet, 286
  xml_node, 303
authcid
  sasl_defaults, 292
authport
  radius_server, 287
authzid
  sasl_defaults, 292

b64enc
  Micelaneous utilities., 124
b64enc_buf
  Micelaneous utilities., 125
basic_auth, 253
  passwd, 253
  user, 253
bio
  ssldata, 296
bl
  ldap_add, 270
  ldap_modify, 276
blist
  bucket_loop, 257
blist_cb
  Hashed bucket linked lists of referenced objects, 46
blist_obj, 254
  data, 254
  hash, 254
  next, 254
  prev, 254
blisthash
  Hashed bucket linked lists of referenced objects, 46
body
  curlbuf, 261
bsize
  curlbuf, 261
bucket
  bucket_loop, 257
bucket_list, 255
  bucketbits, 255
  count, 255
  hash_func, 255
  list, 256
  locks, 256
  version, 256
bucket_list_cnt
  Hashed bucket linked lists of referenced objects, 48
bucket_list_find_key
  Hashed bucket linked lists of referenced objects, 48
bucket_loop, 256
  blist, 257
  bucket, 257
  cur, 257
  cur_hash, 257
  head, 257
  head_hash, 257
  version, 257
bucketbits
  bucket_list, 255
bucketlist_callback
  Hashed bucket linked lists of referenced objects, 49
buf
  ipaddr_req, 268
  iplink_req, 269
buff
  zobj, 307
buffer
  ldap_attrval, 272
Burtle Bob hash algorythim., 219
  final, 220
  HASH_BIG_ENDIAN, 220

HASH_LITTLE_ENDIAN, 221
hashbig, 222
hashlittle, 225
hashlittle2, 228
hashmask, 221
hashsize, 221
hashword, 232
hashword2, 232
JHASH_INITVAL, 221
jenhash, 221
mix, 221
rot, 222

c_type
curlbuf, 261
CURL Url interface., 206
curl_authcb, 207
curl_buf2xml, 208
curl_geturl, 209
curl_newauth, 209
curl_newpost, 210
curl_post, 207
curl_postitem, 210
curl_posturl, 211
curl_progress_func, 207
curl_progress_newdata, 208
curl_progress_pause, 208
curl_setauth_cb, 211
curl_setprogress, 212
curl_ungzip, 212
curlclose, 213
curlinit, 213
url_escape, 214
url_unescape, 214
cat
config_file, 259
cb
nfq_queue, 282
cb_data
radius_session, 289
check_ipv4
IPv4 functions, 146
checkipv6mask
IPv6 functions, 156
checksum
Micelaneous utilities., 125
checksum_add
Micelaneous utilities., 126
children
fwsocket, 265
cidrcnt
IPv4 functions, 146
cidrtosn
IPv4 functions, 146
cleanup
socket_handler, 294
thread_pvt, 297
clearflag
Referenced Lockable Objects, 36
client
socket_handler, 294
unixclient_sockthread, 300
client_func
doxygen/examples/socket.c, 310
close_socket
Network socket interface, 65
closenetlink
Linux network interface functions, 96
cnt
ldap_modreq, 276
ref_obj, 291
xml_node_iter, 304
code
radius_packet, 286
config_cat_callback
INI Style config file Interface, 109
config_catcb
INI Style config file Interface, 109
config_category, 258
entries, 258
name, 258
config_entry, 258
item, 259
value, 259
config_entry_callback
INI Style config file Interface, 110
config_entrycb
INI Style config file Interface, 109
config_file, 259
cat, 259
filename, 260
filepath, 260
config_file_callback
INI Style config file Interface, 110
config_filecb
INI Style config file Interface, 109
connect
socket_handler, 294
Connection Tracking, 243
NFCTRACK_DONE, 244
NF_CTRACK_FLAGS, 244
nf_ctrack_buildct, 244
nf_ctrack_close, 244
nf_ctrack_delete, 245
nf_ctrack_dump, 245
nf_ctrack_endtrace, 245
nf_ctrack_init, 246
nf_ctrack_nat, 246
nf_ctrack_trace, 246
nfct_struct, 243
connex
radius_server, 287
count
bucket_list, 255
ldap_attr, 270
ldap_results, 279
create_bucketlist

Hashed bucket linked lists of referenced objects, 50
create_kernmac
Linux network interface functions, 96
create_kernvlan
Linux network interface functions, 97
create_tun
Linux network interface functions, 98
cred
ldap_simple, 280
ctx
ssldata, 296
cur
bucket_loop, 257
cur_hash
bucket_loop, 257
curl_authcb
CURL Url interface., 207
curl_buf2xml
CURL Url interface., 208
curl_geturl
CURL Url interface., 209
curl_newauth
CURL Url interface., 209
curl_newpost
CURL Url interface., 210
curl_post, 260
CURL Url interface., 207
first, 260
last, 260
curl_postitem
CURL Url interface., 210
curl_posturl
CURL Url interface., 211
curl_progress_func
CURL Url interface., 207
curl_progress_newdata
CURL Url interface., 208
curl_progress_pause
CURL Url interface., 208
curl_setauth_cb
CURL Url interface., 211
curl_setprogress
CURL Url interface., 212
curl_ungzip
CURL Url interface., 212
curlbuf, 261
body, 261
bsize, 261
c_type, 261
header, 261
hsize, 262
curlclose
CURL Url interface., 213
curlinit
CURL Url interface., 213
curpos
xml_node_iter, 304

DTS_OJBREF_CLASS
Referenced Lockable Objects, 36
daddr
pseudohdr, 284
daemonize
Distrotech Application Library, 31
data
blist_obj, 254
nfq_queue, 282
ref_obj, 291
socket_handler, 294
thread_pvt, 297
unixclient_sockthread, 300
unixserv_sockthread, 301
delete_kernmac
Linux network interface functions, 98
delete_kernvlan
Linux network interface functions, 99
destroy
ref_obj, 291
developer
framework_core, 263
Distrotech Application Library, 27
ALLOC_CONST, 29
daemonize, 31
FRAMEWORK_FLAG_DAEMON, 30
FRAMEWORK_FLAG_DAEMONLOCK, 30
FRAMEWORK_FLAG_NOGNU, 30
FRAMEWORK_MAIN, 29
framework_flags, 30
framework_init, 31
framework_mkcore, 32
frameworkfunc, 30
lockpidfile, 33
printgnu, 34
syssighandler, 30
Distrotech Application Library (Todo), 241
dn
ldap_add, 270
ldap_entry, 274
ldap_modify, 276
ldap_simple, 280
dnufn
ldap_entry, 274
doc
xslt_doc, 306
doxygen/dox/buckets.dox, 309
doxygen/dox/examples.dox, 309
doxygen/dox/index.dox, 309
doxygen/dox/main.dox, 309
doxygen/dox/modules.dox, 309
doxygen/dox/refobj.dox, 309
doxygen/dox/sockets.dox, 309
doxygen/dox/sockex.dox, 309
doxygen/dox/thread.dox, 309
doxygen/examples/socket.c, 309, 314
accept_func, 310
client_func, 310
FRAMEWORK_MAIN, 311

server_func, 312
socktest, 312
unixsocktest, 313
dtls_listenssl
SSL socket support, 79
dtlshandltimeout
SSL socket support, 80
dtlstimeout
SSL socket support, 80
dtlsv1_init
SSL socket support, 81
dtsl_serveropts
SSL socket support, 81

email
framework_core, 263
endpoint
unixclient_sockthread, 300
entries
config_category, 258
ldap_results, 279
epre
natmap, 281
eui48to64
IPv6 functions, 157

FRAMEWORK_FLAG_DAEMON
Distrotech Application Library, 30
FRAMEWORK_FLAG_DAEMONLOCK
Distrotech Application Library, 30
FRAMEWORK_FLAG_NOGNU
Distrotech Application Library, 30
FRAMEWORK_MAIN
Distrotech Application Library, 29
doxygen/examples/socket.c, 311
fd
nfq_struct, 283
File utility functions, 160
is_dir, 160
is_exec, 160
is_file, 161
mk_dir, 161
filename
config_file, 260
filepath
config_file, 260
final
Burtle Bob hash algorythim., 220
first
curl_post, 260
ldap_modreq, 277
first_attr
ldap_entry, 274
first_entry
ldap_results, 279
flags
framework_core, 263
fwsocket, 265
nfq_struct, 283
ssldata, 296
thread_pvt, 298
flock
framework_core, 263
framework_core, 262
developer, 263
email, 263
flags, 263
flock, 263
progname, 263
runfile, 263
sa, 263
sig_handler, 263
www, 264
year, 264
framework_flags
Distrotech Application Library, 30
framework_init
Distrotech Application Library, 31
framework_mkcore
Distrotech Application Library, 32
framework_mkthread
Posix thread interface, 58
framework_threadok
Posix thread interface, 59
frameworkfunc
Distrotech Application Library, 30
func
thread_pvt, 298
fwsocket, 264
addr, 265
children, 265
flags, 265
parent, 265
proto, 265
sock, 265
ssl, 266
type, 266

genrand
Micelaneous utilities., 126
get_category_loop
INI Style config file Interface, 110
get_category_next
INI Style config file Interface, 111
get_config_category
INI Style config file Interface, 111
get_config_entry
INI Style config file Interface, 112
get_config_file
INI Style config file Interface, 112
get_iface_index
Linux network interface functions, 99
get_ifinfo
Windows Support, 238
get_ifipaddr
Linux network interface functions, 100
get_ip6_addrprefix
IPv6 functions, 157

getbcaddr
IPv4 functions, 147
getfirstaddr
IPv4 functions, 147
getlastaddr
IPv4 functions, 148
getnetaddr
IPv4 functions, 149
gzinflatebuf
Zlib Interface, 216

h
nfq_struct, 283
HASH_BIG_ENDIAN
Burtle Bob hash algorythim., 220
HASH_LITTLE_ENDIAN
Burtle Bob hash algorythim., 221
hash
blist_obj, 254
hash_func
bucket_list, 255
hashbig
Burtle Bob hash algorythim., 222
Hashed bucket linked lists of referenced objects, 45
addtobucket, 47
blist_cb, 46
blisthash, 46
bucket_list_cnt, 48
bucket_list_find_key, 48
bucketlist_callback, 49
create_bucketlist, 50
init_bucket_loop, 50
next_bucket_loop, 52
remove_bucket_item, 53
remove_bucket_loop, 54
Hashing and digest functions, 132
hashlittle
Burtle Bob hash algorythim., 225
hashlittle2
Burtle Bob hash algorythim., 228
hashmask
Burtle Bob hash algorythim., 221
hashsize
Burtle Bob hash algorythim., 221
hashword
Burtle Bob hash algorythim., 232
hashword2
Burtle Bob hash algorythim., 232
head
bucket_loop, 257
head_hash
bucket_loop, 257
header
curlbuf, 261
hsize
curlbuf, 262

i
ipaddr_req, 268
iplink_req, 269
IP_PROTO_V4
IPv4 and IPv6 functions, 141
IP_PROTO_V6
IPv4 and IPv6 functions, 141
IPV4_SCORE_RESERVED
Linux network interface functions, 95
IPV4_SCORE_ROUTABLE
Linux network interface functions, 95
IPV4_SCORE_ZEROCONF
Linux network interface functions, 95
IPV6_SCORE_RESERVED
Linux network interface functions, 95
IPV6_SCORE_ROUTABLE
Linux network interface functions, 95
IPV6_SCORE_SIXIN4
Linux network interface functions, 95
IPv4 and IPv6 functions
IP_PROTO_V4, 141
IP_PROTO_V6, 141
INI Style config file Interface, 108
config_cat_callback, 109
config_catcb, 109
config_entry_callback, 110
config_entrycb, 109
config_file_callback, 110
config_filecb, 109
get_category_loop, 110
get_category_next, 111
get_config_category, 111
get_config_entry, 112
get_config_file, 112
process_config, 113
unrefconfigfiles, 114
IPv4 and IPv6 functions, 141
inet_lookup, 142
ipversion, 141
packetchecksum, 143
IPv4 functions, 145
check_ipv4, 146
cidrcnt, 146
cidrtosn, 146
getbcaddr, 147
getfirstaddr, 147
getlastaddr, 148
getnetaddr, 149
ipv4checksum, 149
ipv4icmpchecksum, 151
ipv4tcpchecksum, 151
ipv4udpchecksum, 151
packetchecksumv4, 153
reservedip, 153
score_ipv4, 154
IPv6 functions, 156
checkipv6mask, 156
eui48to64, 157
get_ip6_addrprefix, 157
ipv6to4prefix, 158

packetchecksumv6, 158
score_ipv6, 159
IPv6 Nat Mapping, 234
natmap, 234
rfc6296_map, 234
rfc6296_map_add, 235
rfc6296_test, 236
id
radius_connection, 285
radius_packet, 286
radius_server, 287
radius_session, 289
idx
ifinfo, 267
ifaddr
ifinfo, 267
ifdown
Linux network interface functions, 101
ifhwaddr
Linux network interface functions, 101
ifinfo, 266
idx, 267
ifaddr, 267
ipv4addr, 267
ipv6addr, 267
ifrename
Linux network interface functions, 102
ifup
Linux network interface functions, 102
inet_lookup
IPv4 and IPv6 functions, 142
inet_ntop
Windows Support, 239
init_bucket_loop
Hashed bucket linked lists of referenced objects, 50
interface_bind
Linux network interface functions, 103
ipaddr_req, 267
buf, 268
i, 268
n, 268
iplink_req, 268
buf, 269
i, 269
n, 269
ipre
natmap, 281
ipv4_score
Linux network interface functions, 95
ipv4addr
ifinfo, 267
ipv4checksum
IPv4 functions, 149
ipv4icmpchecksum
IPv4 functions, 151
ipv4tcpchecksum
IPv4 functions, 151
ipv4udpchecksum
IPv4 functions, 151
ipv6_score
Linux network interface functions, 95
ipv6addr
ifinfo, 267
ipv6to4prefix
IPv6 functions, 158
ipversion
IPv4 and IPv6 functions, 141
is_dir
File utility functions, 160
is_exec
File utility functions, 160
is_file
File utility functions, 161
is_gzip
Zlib Interface, 217
item
config_entry, 259

JHASH_INITVAL
Burtle Bob hash algorythim., 221
jenhash
Burtle Bob hash algorythim., 221
jointhreads
Posix thread interface, 59

key
xml_node, 303

LDAP_ATTRTYPE_B64
Openldap/SASL Interface, 165
LDAP_ATTRTYPE_CHAR
Openldap/SASL Interface, 165
LDAP_ATTRTYPE_OCTET
Openldap/SASL Interface, 165
LDAP_STARTTLS_ATTEMPT
Openldap/SASL Interface, 165
LDAP_STARTTLS_ENFORCE
Openldap/SASL Interface, 166
LDAP_STARTTLS_NONE
Openldap/SASL Interface, 165
last
curl_post, 260
ldap_modreq, 277
ldap
ldap_conn, 273
ldap_add, 269
bl, 270
dn, 270
Openldap/SASL Interface, 165
ldap_add_attr
Openldap/SASL Interface, 166
ldap_addinit
Openldap/SASL Interface, 166
ldap_attr, 270
count, 270
name, 270
next, 271

prev, 271
vals, 271
ldap_attrtype
Openldap/SASL Interface, 165
ldap_attrval, 271
buffer, 272
len, 272
type, 272
ldap_conn, 272
ldap, 273
limit, 273
Openldap/SASL Interface, 165
sasl, 273
sctrlsp, 273
simple, 273
timelim, 273
uri, 273
ldap_connect
Openldap/SASL Interface, 167
ldap_doadd
Openldap/SASL Interface, 168
ldap_domodify
Openldap/SASL Interface, 168
ldap_entry, 274
attrs, 274
dn, 274
dnufn, 274
first_attr, 274
list, 275
next, 275
prev, 275
rdn, 275
rdncnt, 275
ldap_errmsg
Openldap/SASL Interface, 170
ldap_getattr
Openldap/SASL Interface, 170
ldap_getentry
Openldap/SASL Interface, 172
ldap_mod_add
Openldap/SASL Interface, 172
ldap_mod_addattr
Openldap/SASL Interface, 173
ldap_mod_del
Openldap/SASL Interface, 173
ldap_mod_delattr
Openldap/SASL Interface, 174
ldap_mod_remattr
Openldap/SASL Interface, 174
ldap_mod_rep
Openldap/SASL Interface, 175
ldap_mod_repattr
Openldap/SASL Interface, 175
ldap_modify, 275
bl, 276
dn, 276
Openldap/SASL Interface, 165
ldap_modifyinit
Openldap/SASL Interface, 176
ldap_modreq, 276
attr, 276
cnt, 276
first, 277
last, 277
ldap_modval, 277
next, 277
value, 277
ldap_rdn, 278
name, 278
next, 278
prev, 278
value, 278
ldap_results, 279
count, 279
entries, 279
first_entry, 279
ldap_saslbind
Openldap/SASL Interface, 176
ldap_search_base
Openldap/SASL Interface, 177
ldap_search_one
Openldap/SASL Interface, 179
ldap_search_sub
Openldap/SASL Interface, 180
ldap_simple, 279
cred, 280
dn, 280
ldap_simplebind
Openldap/SASL Interface, 180
ldap_simplerebind
Openldap/SASL Interface, 181
ldap_starttls
Openldap/SASL Interface, 165
ldap_unref_attr
Openldap/SASL Interface, 182
ldap_unref_entry
Openldap/SASL Interface, 183
len
ldap_attrval, 272
pseudohdr, 284
radius_packet, 286
limit
ldap_conn, 273
Linux Netfilter, 242
Linux network interface functions, 94
closenetlink, 96
create_kernmac, 96
create_kernvlan, 97
create_tun, 98
delete_kernmac, 98
delete_kernvlan, 99
get_iface_index, 99
get_ifipaddr, 100
IPV4_SCORE_RESERVED, 95
IPV4_SCORE_ROUTABLE, 95
IPV4_SCORE_ZEROCONF, 95

IPV6_SCORE_RESERVED, 95
IPV6_SCORE_ROUTABLE, 95
IPV6_SCORE_SIXIN4, 95
ifdown, 101
ifhwaddr, 101
ifrename, 102
ifup, 102
interface_bind, 103
ipv4_score, 95
ipv6_score, 95
randhwaddr, 103
set_interface_addr, 104
set_interface_flags, 104
set_interface_ipaddr, 105
set_interface_name, 106
list
bucket_list, 256
ldap_entry, 275
threadcontainer, 299
lock
ref_obj, 291
lockpidfile
Distrotech Application Library, 33
locks
bucket_list, 256
ltrim
Micelaneous utilities., 126

MD5 Hashing and digest functions, 133
md5cmp, 133
md5hmac, 133
md5sum, 134
md5sum2, 134
magic
ref_obj, 291
make_socket
Network socket interface, 67
manager
threadcontainer, 299
mask
natmap, 281
unixserv_sockthread, 301
mcast4_ip
Multicast sockets, 90
mcast6_ip
Multicast sockets, 90
mcast_socket
Multicast sockets, 91
md5cmp
MD5 Hashing and digest functions, 133
md5hmac
MD5 Hashing and digest functions, 133
md5sum
MD5 Hashing and digest functions, 134
md5sum2
MD5 Hashing and digest functions, 134
mech
sasl_defaults, 292
meth
ssldata, 296
Micelaneous utilities., 123
b64enc, 124
b64enc_buf, 125
checksum, 125
checksum_add, 126
genrand, 126
ltrim, 126
rtrim, 128
seedrand, 128
strlenzero, 129
touch, 129
trim, 130
tvtontp64, 130
verifysum, 130
minserver
radius_session, 289
mix
Burtle Bob hash algorythim., 221
mk_dir
File utility functions, 161
Multicast sockets, 90
mcast4_ip, 90
mcast6_ip, 90
mcast_socket, 91

n
ipaddr_req, 268
iplink_req, 269
NFCTRACK_DONE
Connection Tracking, 244
NFQUEUE_DONE
Queue interface, 249
NF_CTRACK_FLAGS
Connection Tracking, 244
NF_QUEUE_FLAGS
Queue interface, 249
name
config_category, 258
ldap_attr, 270
ldap_rdn, 278
radius_server, 288
xml_attr, 302
xml_node, 303
xslt_param, 307
natmap, 280
adji, 281
adjo, 281
epre, 281
ipre, 281
IPv6 Nat Mapping, 234
mask, 281
Network socket interface, 63
accept_socket, 65
close_socket, 65
make_socket, 67
SOCK_FLAG_BIND, 65
SOCK_FLAG_CLOSE, 65
SOCK_FLAG_MCAST, 65

SOCK_FLAG_SSL, 65
SOCK_FLAG_UNIX, 65
sock_flags, 65
sockaddr2ip, 67
sockbind, 68
sockconnect, 68
socketclient, 69
socketread, 69
socketread_d, 70
socketrecv, 64
socketserver, 71
socketwrite, 72
socketwrite_d, 73
tcpbind, 75
tcpconnect, 75
udpbind, 75
udpconnect, 76
new_radpacket
Radius client interface, 120
next
blist_obj, 254
ldap_attr, 271
ldap_entry, 275
ldap_modval, 277
ldap_rdn, 278
next_bucket_loop
Hashed bucket linked lists of referenced objects, 52
nf_ctrack_buildct
Connection Tracking, 244
nf_ctrack_close
Connection Tracking, 244
nf_ctrack_delete
Connection Tracking, 245
nf_ctrack_dump
Connection Tracking, 245
nf_ctrack_endtrace
Connection Tracking, 245
nf_ctrack_init
Connection Tracking, 246
nf_ctrack_nat
Connection Tracking, 246
nf_ctrack_trace
Connection Tracking, 246
nfct_struct
Connection Tracking, 243
nfq
nfq_queue, 282
nfq_data
Queue interface, 248
nfq_queue, 281
cb, 282
data, 282
nfq, 282
num, 282
qh, 282
Queue interface, 248
nfq_struct, 282
fd, 283
flags, 283
h, 283
pf, 283
nfqnl_msg_packet_hdr
Queue interface, 249
nfqueue_attach
Queue interface, 249
nfqueue_cb
Queue interface, 249
nodeptr
xml_node, 303
nodes
xml_search, 305
num
nfq_queue, 282

objalloc
Referenced Lockable Objects, 37
objchar
Referenced Lockable Objects, 38
objcnt
Referenced Lockable Objects, 38
objdestroy
Referenced Lockable Objects, 37
objlock
Referenced Lockable Objects, 40
objref
Referenced Lockable Objects, 40
objsize
Referenced Lockable Objects, 41
objtrylock
Referenced Lockable Objects, 42
objunlock
Referenced Lockable Objects, 42
objunref
Referenced Lockable Objects, 43
olen
radius_session, 289
zobj, 307
Openldap/SASL Interface
LDAP_ATTRTYPE_B64, 165
LDAP_ATTRTYPE_CHAR, 165
LDAP_ATTRTYPE_OCTET, 165
LDAP_STARTTLS_ATTEMPT, 165
LDAP_STARTTLS_ENFORCE, 166
LDAP_STARTTLS_NONE, 165
Openldap/SASL Interface, 163
ldap_add, 165
ldap_add_attr, 166
ldap_addinit, 166
ldap_attrtype, 165
ldap_conn, 165
ldap_connect, 167
ldap_doadd, 168
ldap_domodify, 168
ldap_errmsg, 170
ldap_getattr, 170
ldap_getentry, 172
ldap_mod_add, 172

ldap_mod_addattr, 173
ldap_mod_del, 173
ldap_mod_delattr, 174
ldap_mod_remattr, 174
ldap_mod_rep, 175
ldap_mod_repattr, 175
ldap_modify, 165
ldap_modifyinit, 176
ldap_saslbind, 176
ldap_search_base, 177
ldap_search_one, 179
ldap_search_sub, 180
ldap_simplebind, 180
ldap_simplerebind, 181
ldap_starttls, 165
ldap_unref_attr, 182
ldap_unref_entry, 183

packet
radius_session, 289
packetchecksum
IPv4 and IPv6 functions, 143
packetchecksumv4
IPv4 functions, 153
packetchecksumv6
IPv6 functions, 158
params
xslt_doc, 306
parent
fwsocket, 265
ssldata, 296
passwd
basic_auth, 253
radius_session, 289
sasl_defaults, 293
pf
nfq_struct, 283
Posix thread interface, 55
framework_mkthread, 58
framework_threadok, 59
jointhreads, 59
startthreads, 60
stopthreads, 60
THREAD_OPTION_CANCEL, 57
THREAD_OPTION_JOINABLE, 57
THREAD_OPTION_RETURN, 57
TL_THREAD_CAN_CANCEL, 57
TL_THREAD_DONE, 57
TL_THREAD_JOIN, 57
TL_THREAD_JOINABLE, 57
TL_THREAD_NONE, 57
TL_THREAD_RETURN, 57
TL_THREAD_RUN, 57
TL_THREAD_STOP, 57
thread_can_start, 62
thread_option_flags, 57
thread_signal, 61
threadcleanup, 56
threadfunc, 56
threadopt, 57
threads, 62
threadsighandler, 56
prev
blist_obj, 254
ldap_attr, 271
ldap_entry, 275
ldap_rdn, 278
printgnu
Distrotech Application Library, 34
process_config
INI Style config file Interface, 113
progname
framework_core, 263
proto
fwsocket, 265
pseudohdr, 284
protocol
unixserv_sockthread, 301
pseudohdr, 283
daddr, 284
len, 284
proto, 284
saddr, 284
zero, 284

qh
nfq_queue, 282
Queue interface, 248
NFQUEUE_DONE, 249
NF_QUEUE_FLAGS, 249
nfq_data, 248
nfq_queue, 248
nfqnl_msg_packet_hdr, 249
nfqueue_attach, 249
nfqueue_cb, 249
snprintf_pkt, 250

RAD_CODE_ACCTREQUEST
Radius client interface, 118
RAD_CODE_ACCTRESPONSE
Radius client interface, 118
RAD_CODE_AUTHACCEPT
Radius client interface, 118
RAD_CODE_AUTHCHALLENGE
Radius client interface, 118
RAD_CODE_AUTHREJECT
Radius client interface, 118
RAD_CODE_AUTHREQUEST
Radius client interface, 118
RAD_ATTR_ACCTID
Radius client interface, 116
RAD_ATTR_EAP
Radius client interface, 116
RAD_ATTR_MESSAGE
Radius client interface, 116
RAD_ATTR_NAS_PORT
Radius client interface, 117
RAD_ATTR_PORT_TYPE

Radius client interface, 117
RAD_ATTR_USER_NAME
Radius client interface, 117
RAD_AUTH_HDR_LEN
Radius client interface, 117
RAD_AUTH_TOKEN_LEN
Radius client interface, 117
RAD_MAX_PASS_LEN
Radius client interface, 118
RADIUS_CODE
Radius client interface, 118
REFOBJ_MAGIC
Referenced Lockable Objects, 36
Radius client interface, 115
add_radserver, 119
addradattrint, 119
addradattrip, 119
addradattrstr, 120
new_radpacket, 120
RAD_CODE_ACCTREQUEST, 118
RAD_CODE_ACCTRESPONSE, 118
RAD_CODE_AUTHACCEPT, 118
RAD_CODE_AUTHCHALLENGE, 118
RAD_CODE_AUTHREJECT, 118
RAD_CODE_AUTHREQUEST, 118
RAD_ATTR_ACCTID, 116
RAD_ATTR_EAP, 116
RAD_ATTR_MESSAGE, 116
RAD_ATTR_NAS_PORT, 117
RAD_ATTR_PORT_TYPE, 117
RAD_ATTR_USER_NAME, 117
RAD_AUTH_HDR_LEN, 117
RAD_AUTH_TOKEN_LEN, 117
RAD_MAX_PASS_LEN, 118
RADIUS_CODE, 118
radius_attr_first, 120
radius_attr_next, 121
radius_cb, 118
radius_packet, 118
send_radpacket, 121
radius_attr_first
Radius client interface, 120
radius_attr_next
Radius client interface, 121
radius_cb
Radius client interface, 118
radius_connection, 284
id, 285
server, 285
sessions, 285
socket, 285
radius_packet, 285
attrs, 286
code, 286
id, 286
len, 286
Radius client interface, 118
token, 286
radius_server, 287
acctport, 287
authport, 287
connex, 287
id, 287
name, 288
secret, 288
service, 288
timeout, 288
radius_session, 288
cb_data, 289
id, 289
minserver, 289
olen, 289
packet, 289
passwd, 289
read_cb, 290
request, 290
retries, 290
sent, 290
randhwaddr
Linux network interface functions, 103
rdn
ldap_entry, 275
rdncnt
ldap_entry, 275
read
unixserv_sockthread, 301
read_cb
radius_session, 290
realm
sasl_defaults, 293
ref_obj, 290
cnt, 291
data, 291
destroy, 291
lock, 291
magic, 291
size, 291
Referenced Lockable Objects, 35
clearflag, 36
DTS_OJBREF_CLASS, 36
objalloc, 37
objchar, 38
objcnt, 38
objdestroy, 37
objlock, 40
objref, 40
objsize, 41
objtrylock, 42
objunlock, 42
objunref, 43
REFOBJ_MAGIC, 36
refobj_offset, 37
setflag, 37
testflag, 37
refobj_offset
Referenced Lockable Objects, 37

remove_bucket_item
Hashed bucket linked lists of referenced objects, 53
remove_bucket_loop
Hashed bucket linked lists of referenced objects, 54
request
radius_session, 290
reservedip
IPv4 functions, 153
retries
radius_session, 290
rfc6296_map
IPv6 Nat Mapping, 234
rfc6296_map_add
IPv6 Nat Mapping, 235
rfc6296_test
IPv6 Nat Mapping, 236
rot
Burtle Bob hash algorythim., 222
rtrim
Micelaneous utilities., 128
runfile
framework_core, 263

SOCK_FLAG_BIND
Network socket interface, 65
SOCK_FLAG_CLOSE
Network socket interface, 65
SOCK_FLAG_MCAST
Network socket interface, 65
SOCK_FLAG_SSL
Network socket interface, 65
SOCK_FLAG_UNIX
Network socket interface, 65
SSL socket support
SSL_CLIENT, 78
SSL_DTLSCON, 79
SSL_DTLSV1, 78
SSL_SERVER, 79
SSL_SSLV2, 78
SSL_SSLV3, 78
SSL_TLSV1, 78
SSL_CLIENT
SSL socket support, 78
SSL_DTLSCON
SSL socket support, 79
SSL_DTLSV1
SSL socket support, 78
SSL_SERVER
SSL socket support, 79
SSL_SSLV2
SSL socket support, 78
SSL_SSLV3
SSL socket support, 78
SSL_TLSV1
SSL socket support, 78
SHA1 Hashing and digest functions, 135
sha1cmp, 135
sha1hmac, 135
sha1sum, 136
sha1sum2, 136
SHA2-256Hashing and digest functions, 137
sha256cmp, 137
sha256hmac, 137
sha256sum, 138
sha256sum2, 138
SHA2-512 Hashing and digest functions, 139
sha512cmp, 139
sha512hmac, 139
sha512sum, 140
sha512sum2, 140
SSL socket support, 77
dtls_listenssl, 79
dtlshandltimeout, 80
dtlstimeout, 80
dtlsv1_init, 81
dtsl_serveropts, 81
SSLFLAGS, 78
ssl_shutdown, 82
ssldata, 78
sslstartup, 82
sslv2_init, 82
sslv3_init, 83
startsslclient, 83
tlsaccept, 84
tlsv1_init, 84
SSLFLAGS
SSL socket support, 78
sa
framework_core, 263
sockstruct, 295
sa4
sockstruct, 295
sa6
sockstruct, 295
saddr
pseudohdr, 284
sasl
ldap_conn, 273
sasl_defaults, 292
authcid, 292
authzid, 292
mech, 292
passwd, 293
realm, 293
score_ipv4
IPv4 functions, 154
score_ipv6
IPv6 functions, 159
sctrlsp
ldap_conn, 273
secret
radius_server, 288
seedrand
Micelaneous utilities., 128
send_radpacket
Radius client interface, 121
sent

radius_session, 290
server
radius_connection, 285
server_func
doxygen/examples/socket.c, 312
service
radius_server, 288
sessions
radius_connection, 285
set_interface_addr
Linux network interface functions, 104
set_interface_flags
Linux network interface functions, 104
set_interface_ipaddr
Linux network interface functions, 105
set_interface_name
Linux network interface functions, 106
setflag
Referenced Lockable Objects, 37
sha1cmp
SHA1 Hashing and digest functions, 135
sha1hmac
SHA1 Hashing and digest functions, 135
sha1sum
SHA1 Hashing and digest functions, 136
sha1sum2
SHA1 Hashing and digest functions, 136
sha256cmp
SHA2-256Hashing and digest functions, 137
sha256hmac
SHA2-256Hashing and digest functions, 137
sha256sum
SHA2-256Hashing and digest functions, 138
sha256sum2
SHA2-256Hashing and digest functions, 138
sha512cmp
SHA2-512 Hashing and digest functions, 139
sha512hmac
SHA2-512 Hashing and digest functions, 139
sha512sum
SHA2-512 Hashing and digest functions, 140
sha512sum2
SHA2-512 Hashing and digest functions, 140
sig_handler
framework_core, 263
sighandler
thread_pvt, 298
simple
ldap_conn, 273
size
ref_obj, 291
snprintf_pkt
Queue interface, 250
sock
fwsocket, 265
socket_handler, 294
unixclient_sockthread, 300
unixserv_sockthread, 301
sock_flags
Network socket interface, 65
sockaddr2ip
Network socket interface, 67
sockbind
Network socket interface, 68
sockconnect
Network socket interface, 68
socket
radius_connection, 285
socket_handler, 293
cleanup, 294
client, 294
connect, 294
data, 294
sock, 294
socketclient
Network socket interface, 69
socketread
Network socket interface, 69
socketread_d
Network socket interface, 70
socketrecv
Network socket interface, 64
socketserver
Network socket interface, 71
socketwrite
Network socket interface, 72
socketwrite_d
Network socket interface, 73
sockpath
unixserv_sockthread, 301
sockstruct, 294
sa, 295
sa4, 295
sa6, 295
ss, 295
un, 295
socktest
doxygen/examples/socket.c, 312
src/config.c, 324, 325
src/curl.c, 329, 330
src/fileutil.c, 335, 336
src/include/dtsapp.h, 337, 350
src/interface.c, 358, 360
src/iputil.c, 369, 370
src/libxml2.c, 375, 377
src/libxslt.c, 385, 386
src/lookup3.c, 388, 389
src/main.c, 402, 403
src/nf_ctrack.c, 406, 407
src/nf_queue.c, 410, 411
src/openldap.c, 415, 417
src/radius.c, 434, 436
src/refobj.c, 442, 444
src/rfc6296.c, 451, 452
src/socket.c, 316, 317
src/sslutil.c, 454, 455

src/thread.c, 464, 465
src/unixsock.c, 470, 471
src/util.c, 475, 477
src/winiface.cpp, 481
src/zlib.c, 483
ss
    sockstruct, 295
ssl
    fwsocket, 266
    ssldata, 297
ssl_shutdown
    SSL socket support, 82
ssldata, 295
    bio, 296
    ctx, 296
    flags, 296
    meth, 296
    parent, 296
    ssl, 297
    SSL socket support, 78
sslstartup
    SSL socket support, 82
sslv2_init
    SSL socket support, 82
sslv3_init
    SSL socket support, 83
startsslclient
    SSL socket support, 83
startthreads
    Posix thread interface, 60
stopthreads
    Posix thread interface, 60
strlenzero
    Micelaneous utilities., 129
syssighandler
    Distrotech Application Library, 30

THREAD_OPTION_CANCEL
    Posix thread interface, 57
THREAD_OPTION_JOINABLE
    Posix thread interface, 57
THREAD_OPTION_RETURN
    Posix thread interface, 57
TL_THREAD_CAN_CANCEL
    Posix thread interface, 57
TL_THREAD_DONE
    Posix thread interface, 57
TL_THREAD_JOIN
    Posix thread interface, 57
TL_THREAD_JOINABLE
    Posix thread interface, 57
TL_THREAD_NONE
    Posix thread interface, 57
TL_THREAD_RETURN
    Posix thread interface, 57
TL_THREAD_RUN
    Posix thread interface, 57
TL_THREAD_STOP
    Posix thread interface, 57
tcpbind
    Network socket interface, 75
tcpconnect
    Network socket interface, 75
testflag
    Referenced Lockable Objects, 37
thr
    thread_pvt, 298
thread_can_start
    Posix thread interface, 62
thread_option_flags
    Posix thread interface, 57
thread_pvt, 297
    cleanup, 297
    data, 297
    flags, 298
    func, 298
    sighandler, 298
    thr, 298
thread_signal
    Posix thread interface, 61
threadcleanup
    Posix thread interface, 56
threadcontainer, 298
    list, 299
    manager, 299
threadfunc
    Posix thread interface, 56
threadopt
    Posix thread interface, 57
threads
    Posix thread interface, 62
threadsighandler
    Posix thread interface, 56
timelim
    ldap_conn, 273
timeout
    radius_server, 288
tlsaccept
    SSL socket support, 84
tlsv1_init
    SSL socket support, 84
token
    radius_packet, 286
touch
    Micelaneous utilities., 129
trim
    Micelaneous utilities., 130
tvtontp64
    Micelaneous utilities., 130
type
    fwsocket, 266
    ldap_attrval, 272

udpbind
    Network socket interface, 75
udpconnect
    Network socket interface, 76
un

sockstruct, 295
Unix domain sockets, 86
unixsocket_client, 86
unixsocket_server, 87
unixclient_sockthread, 299
client, 300
data, 300
endpoint, 300
sock, 300
unixserv_sockthread, 300
data, 301
mask, 301
protocol, 301
read, 301
sock, 301
sockpath, 301
unixsocket_client
Unix domain sockets, 86
unixsocket_server
Unix domain sockets, 87
unixsocktest
doxygen/examples/socket.c, 313
unrefconfigfiles
INI Style config file Interface, 114
uri
ldap_conn, 273
url_escape
CURL Url interface., 214
url_unescape
CURL Url interface., 214
user
basic_auth, 253

vals
ldap_attr, 271
value
config_entry, 259
ldap_modval, 277
ldap_rdn, 278
xml_attr, 302
xml_node, 303
xslt_param, 307
verifysum
Micelaneous utilities., 130
version
bucket_list, 256
bucket_loop, 257

Windows Support, 238
get_ifinfo, 238
inet_ntop, 239
www
framework_core, 264

XML Interface, 184
xml_addnode, 186
xml_appendnode, 187
xml_close, 187
xml_createpath, 187
xml_delete, 188
xml_doc, 185
xml_doctobuffer, 190
xml_free_buffer, 190
xml_getattr, 190
xml_getbuffer, 191
xml_getfirstnode, 191
xml_getnextnode, 192
xml_getnode, 193
xml_getnodes, 193
xml_getrootname, 193
xml_getrootnode, 194
xml_init, 194
xml_loadbuf, 194
xml_loaddoc, 195
xml_modify, 195
xml_node, 185
xml_nodecount, 197
xml_savefile, 197
xml_search, 186
xml_setattr, 198
xml_unlink, 198
xml_xpath, 198
XSLT Interface, 200
xslt_addparam, 201
xslt_apply, 202
xslt_apply_buffer, 202
xslt_clearparam, 204
xslt_close, 204
xslt_doc, 200
xslt_init, 205
xslt_open, 205
xml_addnode
XML Interface, 186
xml_appendnode
XML Interface, 187
xml_attr, 302
name, 302
value, 302
xml_close
XML Interface, 187
xml_createpath
XML Interface, 187
xml_delete
XML Interface, 188
xml_doc
XML Interface, 185
xml_doctobuffer
XML Interface, 190
xml_free_buffer
XML Interface, 190
xml_getattr
XML Interface, 190
xml_getbuffer
XML Interface, 191
xml_getfirstnode
XML Interface, 191
xml_getnextnode

XML Interface, 192
xml_getnode
XML Interface, 193
xml_getnodes
XML Interface, 193
xml_getrootname
XML Interface, 193
xml_getrootnode
XML Interface, 194
xml_init
XML Interface, 194
xml_loadbuf
XML Interface, 194
xml_loaddoc
XML Interface, 195
xml_modify
XML Interface, 195
xml_node, 302
attrs, 303
key, 303
name, 303
nodeptr, 303
value, 303
XML Interface, 185
xml_node_iter, 304
cnt, 304
curpos, 304
xsearch, 304
xml_nodecount
XML Interface, 197
xml_savefile
XML Interface, 197
xml_search, 304
nodes, 305
XML Interface, 186
xmldoc, 305
xpathObj, 305
xml_setattr
XML Interface, 198
xml_unlink
XML Interface, 198
xml_xpath
XML Interface, 198
xmldoc
xml_search, 305
xpathObj
xml_search, 305
xsearch
xml_node_iter, 304
xslt_addparam
XSLT Interface, 201
xslt_apply
XSLT Interface, 202
xslt_apply_buffer
XSLT Interface, 202
xslt_clearparam
XSLT Interface, 204
xslt_close
XSLT Interface, 204
xslt_doc, 305
doc, 306
params, 306
XSLT Interface, 200
xslt_init
XSLT Interface, 205
xslt_open
XSLT Interface, 205
xslt_param, 306
name, 307
value, 307

year
framework_core, 264

zcompress
Zlib Interface, 217
zero
pseudohdr, 284
zlen
zobj, 308
Zlib Interface, 216
gzinflatebuf, 216
is_gzip, 217
zcompress, 217
zuncompress, 218
zobj, 307
buff, 307
olen, 307
zlen, 308
zuncompress
Zlib Interface, 218