

DTS Application Library  
0.2.3

Generated by Doxygen 1.8.2

Wed Oct 23 2013 07:33:28



# Contents

<b>1</b>	<b>Distrotech Application Library Manual</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Further information . . . . .	1
1.3	Copyright information. . . . .	1
<b>2</b>	<b>Application Startup</b>	<b>3</b>
2.1	Using helper macro instead of main() . . . . .	3
<b>3</b>	<b>Referenced Lockable Objects</b>	<b>5</b>
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
<b>4</b>	<b>Hashed Bucket Lists</b>	<b>9</b>
4.1	Introduction . . . . .	9
4.2	Usage of hashed bucket lists . . . . .	10
<b>5</b>	<b>Thread Interface</b>	<b>11</b>
5.1	Introduction . . . . .	11
5.2	Creating A Thread . . . . .	11
<b>6</b>	<b>Socket Interface</b>	<b>13</b>
6.1	Introduction . . . . .	13
6.2	SSL Support . . . . .	13
6.3	Socket Creation . . . . .	13
6.4	Starting A Socket . . . . .	13
6.5	Reading/Writing To Sockets . . . . .	14
6.6	Unix Domain Sockets . . . . .	14
6.7	Multicast Sockets . . . . .	14
6.8	Example Code . . . . .	14
<b>7</b>	<b>Socket Example (Echo Server/Client)</b>	<b>15</b>

7.1	Details	15
7.2	Annotation	15
7.3	Code	15
7.4	Output	17
<b>8</b>	<b>Todo List</b>	<b>19</b>
<b>9</b>	<b>Module Index</b>	<b>21</b>
9.1	Modules	21
<b>10</b>	<b>Data Structure Index</b>	<b>23</b>
10.1	Data Structures	23
<b>11</b>	<b>File Index</b>	<b>25</b>
11.1	File List	25
<b>12</b>	<b>Module Documentation</b>	<b>27</b>
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	30
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	36
12.2.1	Detailed Description	37
12.2.2	Macro Definition Documentation	37
12.2.2.1	clearflag	37
12.2.2.2	DTS_OJBREF_CLASS	37
12.2.2.3	REFOBJ_MAGIC	38
12.2.2.4	refobj_offset	38
12.2.2.5	setflag	38
12.2.2.6	testflag	38

12.2.3	Typedef Documentation . . . . .	38
12.2.3.1	objdestroy . . . . .	38
12.2.4	Function Documentation . . . . .	38
12.2.4.1	objalloc . . . . .	38
12.2.4.2	objchar . . . . .	39
12.2.4.3	objcnt . . . . .	40
12.2.4.4	objlock . . . . .	40
12.2.4.5	objref . . . . .	41
12.2.4.6	objsize . . . . .	42
12.2.4.7	objtrylock . . . . .	42
12.2.4.8	objunlock . . . . .	43
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 . . . . .	47
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 . . . . .	50
12.3.3.5	create_bucketlist . . . . .	50
12.3.3.6	init_bucket_loop . . . . .	51
12.3.3.7	next_bucket_loop . . . . .	51
12.3.3.8	remove_bucket_item . . . . .	52
12.3.3.9	remove_bucket_loop . . . . .	53
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	66
12.5.4.4	sockaddr2ip	67
12.5.4.5	sockbind	67
12.5.4.6	sockconnect	68
12.5.4.7	socketclient	68
12.5.4.8	socketread	69
12.5.4.9	socketread_d	69
12.5.4.10	socketserver	71
12.5.4.11	socketwrite	71
12.5.4.12	socketwrite_d	72
12.5.4.13	tcpbind	74
12.5.4.14	tcpconnect	74
12.5.4.15	udpbind	75
12.5.4.16	udpconnect	75
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	<a href="#">dtlstimeout</a>	80
12.6.5.4	<a href="#">dtlsv1_init</a>	81
12.6.5.5	<a href="#">dtls_serveropts</a>	81
12.6.5.6	<a href="#">ssl_shutdown</a>	82
12.6.5.7	<a href="#">sslstartup</a>	82
12.6.5.8	<a href="#">sslv2_init</a>	83
12.6.5.9	<a href="#">sslv3_init</a>	83
12.6.5.10	<a href="#">startsslclient</a>	84
12.6.5.11	<a href="#">tlsaccept</a>	84
12.6.5.12	<a href="#">tlsv1_init</a>	85
12.7	<a href="#">Unix domain sockets</a>	86
12.7.1	<a href="#">Detailed Description</a>	86
12.7.2	<a href="#">Function Documentation</a>	86
12.7.2.1	<a href="#">unixsocket_client</a>	86
12.7.2.2	<a href="#">unixsocket_server</a>	88
12.8	<a href="#">Multicast sockets</a>	89
12.8.1	<a href="#">Detailed Description</a>	89
12.8.2	<a href="#">Function Documentation</a>	89
12.8.2.1	<a href="#">mcast4_ip</a>	89
12.8.2.2	<a href="#">mcast6_ip</a>	89
12.8.2.3	<a href="#">mcast_socket</a>	90
12.9	<a href="#">Linux network interface functions</a>	93
12.9.1	<a href="#">Detailed Description</a>	94
12.9.2	<a href="#">Enumeration Type Documentation</a>	94
12.9.2.1	<a href="#">ipv4_score</a>	94
12.9.2.2	<a href="#">ipv6_score</a>	94
12.9.3	<a href="#">Function Documentation</a>	95
12.9.3.1	<a href="#">closenetwork</a>	95
12.9.3.2	<a href="#">create_kernmac</a>	95
12.9.3.3	<a href="#">create_kernvlan</a>	96
12.9.3.4	<a href="#">create_tun</a>	97
12.9.3.5	<a href="#">delete_kernmac</a>	98
12.9.3.6	<a href="#">delete_kernvlan</a>	98
12.9.3.7	<a href="#">get_iface_index</a>	98
12.9.3.8	<a href="#">get_ifipaddr</a>	99
12.9.3.9	<a href="#">ifdown</a>	100
12.9.3.10	<a href="#">ifhwaddr</a>	100
12.9.3.11	<a href="#">ifrename</a>	101
12.9.3.12	<a href="#">ifup</a>	101
12.9.3.13	<a href="#">interface_bind</a>	102

12.9.3.14 randhwaddr . . . . .	103
12.9.3.15 set_interface_addr . . . . .	103
12.9.3.16 set_interface_flags . . . . .	104
12.9.3.17 set_interface_ipaddr . . . . .	104
12.9.3.18 set_interface_name . . . . .	105
12.10INI Style config file Interface . . . . .	107
12.10.1 Detailed Description . . . . .	108
12.10.2 Typedef Documentation . . . . .	108
12.10.2.1 config_catchb . . . . .	108
12.10.2.2 config_entrycb . . . . .	108
12.10.2.3 config_filecb . . . . .	108
12.10.3 Function Documentation . . . . .	108
12.10.3.1 config_cat_callback . . . . .	108
12.10.3.2 config_entry_callback . . . . .	109
12.10.3.3 config_file_callback . . . . .	109
12.10.3.4 get_category_loop . . . . .	109
12.10.3.5 get_category_next . . . . .	110
12.10.3.6 get_config_category . . . . .	110
12.10.3.7 get_config_entry . . . . .	111
12.10.3.8 get_config_file . . . . .	112
12.10.3.9 process_config . . . . .	112
12.10.3.10unrefconfigfiles . . . . .	113
12.11 Radius client interface . . . . .	114
12.11.1 Detailed Description . . . . .	115
12.11.2 Macro Definition Documentation . . . . .	115
12.11.2.1 RAD_ATTR_ACCTID . . . . .	115
12.11.2.2 RAD_ATTR_EAP . . . . .	115
12.11.2.3 RAD_ATTR_MESSAGE . . . . .	116
12.11.2.4 RAD_ATTR_NAS_IP_ADDR . . . . .	116
12.11.2.5 RAD_ATTR_NAS_PORT . . . . .	116
12.11.2.6 RAD_ATTR_PORT_TYPE . . . . .	116
12.11.2.7 RAD_ATTR_SERVICE_TYPE . . . . .	116
12.11.2.8 RAD_ATTR_USER_NAME . . . . .	116
12.11.2.9 RAD_ATTR_USER_PASSWORD . . . . .	116
12.11.2.10RAD_AUTH_HDR_LEN . . . . .	116
12.11.2.11RAD_AUTH_PACKET_LEN . . . . .	116
12.11.2.12RAD_AUTH_TOKEN_LEN . . . . .	117
12.11.2.13RAD_MAX_PASS_LEN . . . . .	117
12.11.3 Typedef Documentation . . . . .	117
12.11.3.1 radius_cb . . . . .	117



12.11.3.2 radius_packet . . . . .	117
12.11.4 Enumeration Type Documentation . . . . .	117
12.11.4.1 RADIUS_CODE . . . . .	117
12.11.5 Function Documentation . . . . .	118
12.11.5.1 add_radserver . . . . .	118
12.11.5.2 addradattrint . . . . .	118
12.11.5.3 addradattrip . . . . .	118
12.11.5.4 addradattrstr . . . . .	119
12.11.5.5 new_radpacket . . . . .	119
12.11.5.6 radius_attr_first . . . . .	120
12.11.5.7 radius_attr_next . . . . .	120
12.11.5.8 send_radpacket . . . . .	120
12.12 Miscellaneous utilities. . . . .	122
12.12.1 Detailed Description . . . . .	122
12.12.2 Function Documentation . . . . .	123
12.12.2.1 b64enc . . . . .	123
12.12.2.2 b64enc_buf . . . . .	123
12.12.2.3 checksum . . . . .	124
12.12.2.4 checksum_add . . . . .	124
12.12.2.5 genrand . . . . .	124
12.12.2.6 ltrim . . . . .	125
12.12.2.7 rtrim . . . . .	125
12.12.2.8 seedrand . . . . .	126
12.12.2.9 strlenzero . . . . .	126
12.12.2.10 touch . . . . .	126
12.12.2.11 ltrim . . . . .	127
12.12.2.12 vtontp64 . . . . .	127
12.12.2.13 verifysum . . . . .	128
12.13 Hashing and digest functions . . . . .	129
12.13.1 Detailed Description . . . . .	129
12.14 MD5 Hashing and digest functions . . . . .	130
12.14.1 Detailed Description . . . . .	130
12.14.2 Function Documentation . . . . .	130
12.14.2.1 md5cmp . . . . .	130
12.14.2.2 md5hmac . . . . .	130
12.14.2.3 md5sum . . . . .	131
12.14.2.4 md5sum2 . . . . .	131
12.15 SHA1 Hashing and digest functions . . . . .	132
12.15.1 Detailed Description . . . . .	132
12.15.2 Function Documentation . . . . .	132

12.15.2.1 sha1cmp . . . . .	132
12.15.2.2 sha1hmac . . . . .	132
12.15.2.3 sha1sum . . . . .	133
12.15.2.4 sha1sum2 . . . . .	133
12.16SHA2-256Hashing and digest functions . . . . .	134
12.16.1 Detailed Description . . . . .	134
12.16.2 Function Documentation . . . . .	134
12.16.2.1 sha256cmp . . . . .	134
12.16.2.2 sha256hmac . . . . .	134
12.16.2.3 sha256sum . . . . .	135
12.16.2.4 sha256sum2 . . . . .	135
12.17SHA2-512 Hashing and digest functions . . . . .	136
12.17.1 Detailed Description . . . . .	136
12.17.2 Function Documentation . . . . .	136
12.17.2.1 sha512cmp . . . . .	136
12.17.2.2 sha512hmac . . . . .	136
12.17.2.3 sha512sum . . . . .	137
12.17.2.4 sha512sum2 . . . . .	137
12.18IPv4 and IPv6 functions . . . . .	138
12.18.1 Detailed Description . . . . .	138
12.18.2 Enumeration Type Documentation . . . . .	138
12.18.2.1 ipversion . . . . .	138
12.18.3 Function Documentation . . . . .	139
12.18.3.1 inet_lookup . . . . .	139
12.18.3.2 packetchecksum . . . . .	139
12.19IPv4 functions . . . . .	141
12.19.1 Detailed Description . . . . .	141
12.19.2 Function Documentation . . . . .	142
12.19.2.1 check_ipv4 . . . . .	142
12.19.2.2 cidrcnt . . . . .	142
12.19.2.3 cidrtosn . . . . .	143
12.19.2.4 getbcaddr . . . . .	143
12.19.2.5 getfirstaddr . . . . .	144
12.19.2.6 getlastaddr . . . . .	144
12.19.2.7 getnetaddr . . . . .	145
12.19.2.8 ipv4checksum . . . . .	146
12.19.2.9 ipv4icmpchecksum . . . . .	146
12.19.2.10ipv4tcpchecksum . . . . .	146
12.19.2.11ipv4udpchecksum . . . . .	147
12.19.2.12packetchecksumv4 . . . . .	147

12.19.2.13reservedip . . . . .	148
12.19.2.14score_ipv4 . . . . .	149
12.20IPv6 functions . . . . .	150
12.20.1 Detailed Description . . . . .	150
12.20.2 Function Documentation . . . . .	150
12.20.2.1 checkipv6mask . . . . .	150
12.20.2.2 eui48to64 . . . . .	151
12.20.2.3 get_ip6_addrprefix . . . . .	151
12.20.2.4 ipv6to4prefix . . . . .	152
12.20.2.5 packetchecksumv6 . . . . .	152
12.20.2.6 score_ipv6 . . . . .	153
12.21 File utility functions . . . . .	154
12.21.1 Detailed Description . . . . .	154
12.21.2 Function Documentation . . . . .	154
12.21.2.1 is_dir . . . . .	154
12.21.2.2 is_exec . . . . .	154
12.21.2.3 is_file . . . . .	155
12.21.2.4 mk_dir . . . . .	155
12.22Openldap/SASL Interface . . . . .	157
12.22.1 Detailed Description . . . . .	159
12.22.2 Typedef Documentation . . . . .	159
12.22.2.1 ldap_add . . . . .	159
12.22.2.2 ldap_conn . . . . .	159
12.22.2.3 ldap_modify . . . . .	159
12.22.3 Enumeration Type Documentation . . . . .	159
12.22.3.1 ldap_attrtype . . . . .	159
12.22.3.2 ldap_starttls . . . . .	159
12.22.4 Function Documentation . . . . .	160
12.22.4.1 ldap_add_attr . . . . .	160
12.22.4.2 ldap_addinit . . . . .	160
12.22.4.3 ldap_connect . . . . .	161
12.22.4.4 ldap_doaddd . . . . .	162
12.22.4.5 ldap_domodify . . . . .	163
12.22.4.6 ldap_errmsg . . . . .	163
12.22.4.7 ldap_getattr . . . . .	164
12.22.4.8 ldap_getentry . . . . .	164
12.22.4.9 ldap_mod_add . . . . .	165
12.22.4.10ldap_mod_addattr . . . . .	165
12.22.4.11ldap_mod_del . . . . .	166
12.22.4.12dap_mod_delattr . . . . .	166

12.22.4.13	dap_mod_rematrr	167
12.22.4.14	dap_mod_rep	167
12.22.4.15	dap_mod_repattr	168
12.22.4.16	dap_modifyinit	169
12.22.4.17	dap_saslbind	169
12.22.4.18	dap_search_base	170
12.22.4.19	dap_search_one	171
12.22.4.20	dap_search_sub	172
12.22.4.21	dap_simplebind	172
12.22.4.22	dap_simplerebind	173
12.22.4.23	dap_unref_attr	174
12.22.4.24	dap_unref_entry	175
12.23	XML Interface	176
12.23.1	Detailed Description	177
12.23.2	Typedef Documentation	177
12.23.2.1	xml_doc	177
12.23.2.2	xml_node	178
12.23.2.3	xml_search	178
12.23.3	Function Documentation	178
12.23.3.1	xml_addnode	178
12.23.3.2	xml_appendnode	179
12.23.3.3	xml_close	179
12.23.3.4	xml_createpath	179
12.23.3.5	xml_delete	181
12.23.3.6	xml_doctobuffer	181
12.23.3.7	xml_free_buffer	181
12.23.3.8	xml_getattr	182
12.23.3.9	xml_getbuffer	182
12.23.3.10	xml_getfirstnode	183
12.23.3.11	xml_getnextnode	183
12.23.3.12	xml_getnode	184
12.23.3.13	xml_getnodes	184
12.23.3.14	xml_getrootname	185
12.23.3.15	xml_getrootnode	185
12.23.3.16	xml_init	186
12.23.3.17	xml_loadbuf	186
12.23.3.18	xml_loaddoc	187
12.23.3.19	xml_modify	187
12.23.3.20	xml_nodecount	188
12.23.3.21	xml_savefile	188

12.23.3.22xml_setattr . . . . .	188
12.23.3.23xml_unlink . . . . .	189
12.23.3.24xml_xpath . . . . .	189
12.24XSLT Interface . . . . .	191
12.24.1 Detailed Description . . . . .	191
12.24.2 Typedef Documentation . . . . .	191
12.24.2.1 xslt_doc . . . . .	191
12.24.3 Function Documentation . . . . .	192
12.24.3.1 xslt_addparam . . . . .	192
12.24.3.2 xslt_apply . . . . .	192
12.24.3.3 xslt_apply_buffer . . . . .	193
12.24.3.4 xslt_clearparam . . . . .	193
12.24.3.5 xslt_close . . . . .	194
12.24.3.6 xslt_init . . . . .	194
12.24.3.7 xslt_open . . . . .	194
12.25CURL Url interface. . . . .	196
12.25.1 Detailed Description . . . . .	197
12.25.2 Typedef Documentation . . . . .	197
12.25.2.1 curl_authcb . . . . .	197
12.25.2.2 curl_post . . . . .	197
12.25.2.3 curl_progress_func . . . . .	197
12.25.2.4 curl_progress_newdata . . . . .	198
12.25.2.5 curl_progress_pause . . . . .	198
12.25.3 Function Documentation . . . . .	198
12.25.3.1 curl_buf2xml . . . . .	198
12.25.3.2 curl_geturl . . . . .	199
12.25.3.3 curl_newauth . . . . .	199
12.25.3.4 curl_newpost . . . . .	200
12.25.3.5 curl_postitem . . . . .	200
12.25.3.6 curl_posturl . . . . .	201
12.25.3.7 curl_setauth_cb . . . . .	201
12.25.3.8 curl_setprogress . . . . .	202
12.25.3.9 curl_ungzip . . . . .	202
12.25.3.10curlclose . . . . .	203
12.25.3.11curlinit . . . . .	203
12.25.3.12url_escape . . . . .	204
12.25.3.13url_unescape . . . . .	204
12.26Zlib Interface . . . . .	206
12.26.1 Detailed Description . . . . .	206
12.26.2 Function Documentation . . . . .	206

12.26.2.1 gzinflatebuf . . . . .	206
12.26.2.2 is_gzip . . . . .	207
12.26.2.3 zcompress . . . . .	207
12.26.2.4 zuncompress . . . . .	208
12.27 Burtle Bob hash algorithim. . . . .	209
12.27.1 Detailed Description . . . . .	209
12.27.2 Macro Definition Documentation . . . . .	210
12.27.2.1 final . . . . .	210
12.27.2.2 HASH_BIG_ENDIAN . . . . .	211
12.27.2.3 HASH_LITTLE_ENDIAN . . . . .	211
12.27.2.4 hashmask . . . . .	211
12.27.2.5 hashsize . . . . .	211
12.27.2.6 jenkinshash . . . . .	211
12.27.2.7 JHASH_INITVAL . . . . .	211
12.27.2.8 mix . . . . .	211
12.27.2.9 rot . . . . .	212
12.27.3 Function Documentation . . . . .	212
12.27.3.1 hashbig . . . . .	212
12.27.3.2 hashlittle . . . . .	215
12.27.3.3 hashlittle2 . . . . .	219
12.27.3.4 hashword . . . . .	222
12.27.3.5 hashword2 . . . . .	223
12.28 IPv6 Nat Mapping . . . . .	225
12.28.1 Detailed Description . . . . .	225
12.28.2 Typedef Documentation . . . . .	225
12.28.2.1 natmap . . . . .	225
12.28.3 Function Documentation . . . . .	225
12.28.3.1 rfc6296_map . . . . .	225
12.28.3.2 rfc6296_map_add . . . . .	226
12.28.3.3 rfc6296_test . . . . .	227
12.29 Windows Support . . . . .	229
12.29.1 Detailed Description . . . . .	229
12.29.2 Function Documentation . . . . .	229
12.29.2.1 get_ifinfo . . . . .	229
12.29.2.2 inet_ntop . . . . .	230
12.30 Distrotech Application Library (Todo) . . . . .	232
12.30.1 Detailed Description . . . . .	232
12.31 Linux Netfilter . . . . .	233
12.31.1 Detailed Description . . . . .	233
12.32 Connection Tracking . . . . .	234

12.32.1 Detailed Description . . . . .	234
12.32.2 Typedef Documentation . . . . .	234
12.32.2.1 nfct_struct . . . . .	234
12.32.3 Enumeration Type Documentation . . . . .	235
12.32.3.1 NF_CTRACK_FLAGS . . . . .	235
12.32.4 Function Documentation . . . . .	235
12.32.4.1 nf_ctrack_buildct . . . . .	235
12.32.4.2 nf_ctrack_close . . . . .	235
12.32.4.3 nf_ctrack_delete . . . . .	236
12.32.4.4 nf_ctrack_dump . . . . .	236
12.32.4.5 nf_ctrack_endtrace . . . . .	236
12.32.4.6 nf_ctrack_init . . . . .	237
12.32.4.7 nf_ctrack_nat . . . . .	237
12.32.4.8 nf_ctrack_trace . . . . .	238
12.33 Queue interface . . . . .	239
12.33.1 Detailed Description . . . . .	239
12.33.2 Typedef Documentation . . . . .	239
12.33.2.1 nfq_data . . . . .	239
12.33.2.2 nfq_queue . . . . .	240
12.33.2.3 nfqnl_msg_packet_hdr . . . . .	240
12.33.2.4 nfqueue_cb . . . . .	240
12.33.3 Enumeration Type Documentation . . . . .	240
12.33.3.1 NF_QUEUE_FLAGS . . . . .	240
12.33.4 Function Documentation . . . . .	240
12.33.4.1 nfqueue_attach . . . . .	240
12.33.4.2 snprintf_pkt . . . . .	241
<b>13 Data Structure Documentation</b>	<b>243</b>
13.1 basic_auth Struct Reference . . . . .	243
13.1.1 Detailed Description . . . . .	243
13.1.2 Field Documentation . . . . .	243
13.1.2.1 passwd . . . . .	243
13.1.2.2 user . . . . .	243
13.2 blist_obj Struct Reference . . . . .	244
13.2.1 Detailed Description . . . . .	244
13.2.2 Field Documentation . . . . .	244
13.2.2.1 data . . . . .	244
13.2.2.2 hash . . . . .	244
13.2.2.3 next . . . . .	244
13.2.2.4 prev . . . . .	245

13.3 bucket_list Struct Reference . . . . .	245
13.3.1 Detailed Description . . . . .	245
13.3.2 Field Documentation . . . . .	245
13.3.2.1 bucketbits . . . . .	245
13.3.2.2 count . . . . .	245
13.3.2.3 hash_func . . . . .	246
13.3.2.4 list . . . . .	246
13.3.2.5 locks . . . . .	246
13.3.2.6 version . . . . .	246
13.4 bucket_loop Struct Reference . . . . .	246
13.4.1 Detailed Description . . . . .	247
13.4.2 Field Documentation . . . . .	247
13.4.2.1 blist . . . . .	247
13.4.2.2 bucket . . . . .	247
13.4.2.3 cur . . . . .	247
13.4.2.4 cur_hash . . . . .	247
13.4.2.5 head . . . . .	247
13.4.2.6 head_hash . . . . .	247
13.4.2.7 version . . . . .	248
13.5 config_category Struct Reference . . . . .	248
13.5.1 Detailed Description . . . . .	248
13.5.2 Field Documentation . . . . .	248
13.5.2.1 entries . . . . .	248
13.5.2.2 name . . . . .	248
13.6 config_entry Struct Reference . . . . .	248
13.6.1 Detailed Description . . . . .	249
13.6.2 Field Documentation . . . . .	249
13.6.2.1 item . . . . .	249
13.6.2.2 value . . . . .	249
13.7 config_file Struct Reference . . . . .	249
13.7.1 Detailed Description . . . . .	249
13.7.2 Field Documentation . . . . .	249
13.7.2.1 cat . . . . .	249
13.7.2.2 filename . . . . .	250
13.7.2.3 filepath . . . . .	250
13.8 curl_post Struct Reference . . . . .	250
13.8.1 Detailed Description . . . . .	250
13.8.2 Field Documentation . . . . .	250
13.8.2.1 first . . . . .	250
13.8.2.2 last . . . . .	250



13.9 curlbuf Struct Reference . . . . .	251
13.9.1 Detailed Description . . . . .	251
13.9.2 Field Documentation . . . . .	251
13.9.2.1 body . . . . .	251
13.9.2.2 bsize . . . . .	251
13.9.2.3 c_type . . . . .	251
13.9.2.4 header . . . . .	252
13.9.2.5 hsize . . . . .	252
13.10framework_core Struct Reference . . . . .	252
13.10.1 Detailed Description . . . . .	252
13.10.2 Field Documentation . . . . .	253
13.10.2.1 developer . . . . .	253
13.10.2.2 email . . . . .	253
13.10.2.3 flags . . . . .	253
13.10.2.4 flock . . . . .	253
13.10.2.5 progname . . . . .	253
13.10.2.6 runfile . . . . .	253
13.10.2.7 sa . . . . .	253
13.10.2.8 sig_handler . . . . .	254
13.10.2.9 www . . . . .	254
13.10.2.10year . . . . .	254
13.11fwsocket Struct Reference . . . . .	254
13.11.1 Detailed Description . . . . .	255
13.11.2 Field Documentation . . . . .	255
13.11.2.1 addr . . . . .	255
13.11.2.2 children . . . . .	255
13.11.2.3 flags . . . . .	255
13.11.2.4 parent . . . . .	255
13.11.2.5 proto . . . . .	255
13.11.2.6 sock . . . . .	256
13.11.2.7 ssl . . . . .	256
13.11.2.8 type . . . . .	256
13.12ifinfo Struct Reference . . . . .	256
13.12.1 Detailed Description . . . . .	257
13.12.2 Field Documentation . . . . .	257
13.12.2.1 idx . . . . .	257
13.12.2.2 ifaddr . . . . .	257
13.12.2.3 ipv4addr . . . . .	257
13.12.2.4 ipv6addr . . . . .	257
13.13ipaddr_req Struct Reference . . . . .	257

13.13.1 Detailed Description . . . . .	258
13.13.2 Field Documentation . . . . .	258
13.13.2.1 buf . . . . .	258
13.13.2.2 i . . . . .	258
13.13.2.3 n . . . . .	258
13.14iplink_req Struct Reference . . . . .	258
13.14.1 Detailed Description . . . . .	259
13.14.2 Field Documentation . . . . .	259
13.14.2.1 buf . . . . .	259
13.14.2.2 i . . . . .	259
13.14.2.3 n . . . . .	259
13.15ldap_add Struct Reference . . . . .	259
13.15.1 Detailed Description . . . . .	259
13.15.2 Field Documentation . . . . .	260
13.15.2.1 bl . . . . .	260
13.15.2.2 dn . . . . .	260
13.16ldap_attr Struct Reference . . . . .	260
13.16.1 Detailed Description . . . . .	260
13.16.2 Field Documentation . . . . .	260
13.16.2.1 count . . . . .	260
13.16.2.2 name . . . . .	261
13.16.2.3 next . . . . .	261
13.16.2.4 prev . . . . .	261
13.16.2.5 vals . . . . .	261
13.17ldap_attrval Struct Reference . . . . .	261
13.17.1 Detailed Description . . . . .	261
13.17.2 Field Documentation . . . . .	262
13.17.2.1 buffer . . . . .	262
13.17.2.2 len . . . . .	262
13.17.2.3 type . . . . .	262
13.18ldap_conn Struct Reference . . . . .	262
13.18.1 Detailed Description . . . . .	262
13.18.2 Field Documentation . . . . .	263
13.18.2.1 ldap . . . . .	263
13.18.2.2 limit . . . . .	263
13.18.2.3 sasl . . . . .	263
13.18.2.4 sctrlsp . . . . .	263
13.18.2.5 simple . . . . .	263
13.18.2.6 timelim . . . . .	263
13.18.2.7 uri . . . . .	263

13.19ldap_entry Struct Reference . . . . .	264
13.19.1 Detailed Description . . . . .	264
13.19.2 Field Documentation . . . . .	264
13.19.2.1 attrs . . . . .	264
13.19.2.2 dn . . . . .	264
13.19.2.3 dnufn . . . . .	265
13.19.2.4 first_attr . . . . .	265
13.19.2.5 list . . . . .	265
13.19.2.6 next . . . . .	265
13.19.2.7 prev . . . . .	265
13.19.2.8 rdn . . . . .	265
13.19.2.9 rdncnt . . . . .	265
13.20ldap_modify Struct Reference . . . . .	265
13.20.1 Detailed Description . . . . .	266
13.20.2 Field Documentation . . . . .	266
13.20.2.1 bl . . . . .	266
13.20.2.2 dn . . . . .	266
13.21ldap_modreq Struct Reference . . . . .	266
13.21.1 Detailed Description . . . . .	266
13.21.2 Field Documentation . . . . .	267
13.21.2.1 attr . . . . .	267
13.21.2.2 cnt . . . . .	267
13.21.2.3 first . . . . .	267
13.21.2.4 last . . . . .	267
13.22ldap_modval Struct Reference . . . . .	267
13.22.1 Detailed Description . . . . .	267
13.22.2 Field Documentation . . . . .	267
13.22.2.1 next . . . . .	267
13.22.2.2 value . . . . .	268
13.23ldap_rdn Struct Reference . . . . .	268
13.23.1 Detailed Description . . . . .	268
13.23.2 Field Documentation . . . . .	268
13.23.2.1 name . . . . .	268
13.23.2.2 next . . . . .	268
13.23.2.3 prev . . . . .	268
13.23.2.4 value . . . . .	269
13.24ldap_results Struct Reference . . . . .	269
13.24.1 Detailed Description . . . . .	269
13.24.2 Field Documentation . . . . .	269
13.24.2.1 count . . . . .	269

13.24.2.2 entries . . . . .	269
13.24.2.3 first_entry . . . . .	269
13.25ldap_simple Struct Reference . . . . .	270
13.25.1 Detailed Description . . . . .	270
13.25.2 Field Documentation . . . . .	270
13.25.2.1 cred . . . . .	270
13.25.2.2 dn . . . . .	270
13.26natmap Struct Reference . . . . .	270
13.26.1 Detailed Description . . . . .	271
13.26.2 Field Documentation . . . . .	271
13.26.2.1 adjl . . . . .	271
13.26.2.2 adjo . . . . .	271
13.26.2.3 epre . . . . .	271
13.26.2.4 ipre . . . . .	271
13.26.2.5 mask . . . . .	271
13.27nfq_queue Struct Reference . . . . .	272
13.27.1 Detailed Description . . . . .	272
13.27.2 Field Documentation . . . . .	272
13.27.2.1 cb . . . . .	272
13.27.2.2 data . . . . .	272
13.27.2.3 nfq . . . . .	272
13.27.2.4 num . . . . .	272
13.27.2.5 qh . . . . .	272
13.28nfq_struct Struct Reference . . . . .	272
13.28.1 Detailed Description . . . . .	273
13.28.2 Field Documentation . . . . .	273
13.28.2.1 fd . . . . .	273
13.28.2.2 flags . . . . .	273
13.28.2.3 h . . . . .	273
13.28.2.4 pf . . . . .	273
13.29pseudohdr Struct Reference . . . . .	273
13.29.1 Detailed Description . . . . .	274
13.29.2 Field Documentation . . . . .	274
13.29.2.1 daddr . . . . .	274
13.29.2.2 len . . . . .	274
13.29.2.3 proto . . . . .	274
13.29.2.4 saddr . . . . .	274
13.29.2.5 zero . . . . .	274
13.30radius_connection Struct Reference . . . . .	274
13.30.1 Detailed Description . . . . .	275

13.30.2 Field Documentation . . . . .	275
13.30.2.1 id . . . . .	275
13.30.2.2 server . . . . .	275
13.30.2.3 sessions . . . . .	275
13.30.2.4 socket . . . . .	275
13.31 radius_packet Struct Reference . . . . .	275
13.31.1 Detailed Description . . . . .	276
13.31.2 Field Documentation . . . . .	276
13.31.2.1 attrs . . . . .	276
13.31.2.2 code . . . . .	276
13.31.2.3 id . . . . .	276
13.31.2.4 len . . . . .	276
13.31.2.5 token . . . . .	277
13.32 radius_server Struct Reference . . . . .	277
13.32.1 Detailed Description . . . . .	277
13.32.2 Field Documentation . . . . .	277
13.32.2.1 acctport . . . . .	277
13.32.2.2 authport . . . . .	278
13.32.2.3 connex . . . . .	278
13.32.2.4 id . . . . .	278
13.32.2.5 name . . . . .	278
13.32.2.6 secret . . . . .	278
13.32.2.7 service . . . . .	278
13.32.2.8 timeout . . . . .	278
13.33 radius_session Struct Reference . . . . .	279
13.33.1 Detailed Description . . . . .	279
13.33.2 Field Documentation . . . . .	279
13.33.2.1 cb_data . . . . .	279
13.33.2.2 id . . . . .	279
13.33.2.3 minserver . . . . .	279
13.33.2.4 olen . . . . .	280
13.33.2.5 packet . . . . .	280
13.33.2.6 passwd . . . . .	280
13.33.2.7 read_cb . . . . .	280
13.33.2.8 request . . . . .	280
13.33.2.9 retries . . . . .	280
13.33.2.10 sent . . . . .	280
13.34 ref_obj Struct Reference . . . . .	280
13.34.1 Detailed Description . . . . .	281
13.34.2 Field Documentation . . . . .	281

13.34.2.1 cnt . . . . .	281
13.34.2.2 data . . . . .	281
13.34.2.3 destroy . . . . .	281
13.34.2.4 lock . . . . .	281
13.34.2.5 magic . . . . .	281
13.34.2.6 size . . . . .	282
13.35sasl_defaults Struct Reference . . . . .	282
13.35.1 Detailed Description . . . . .	282
13.35.2 Field Documentation . . . . .	282
13.35.2.1 authcid . . . . .	282
13.35.2.2 authzid . . . . .	283
13.35.2.3 mech . . . . .	283
13.35.2.4 passwd . . . . .	283
13.35.2.5 realm . . . . .	283
13.36socket_handler Struct Reference . . . . .	283
13.36.1 Detailed Description . . . . .	284
13.36.2 Field Documentation . . . . .	284
13.36.2.1 cleanup . . . . .	284
13.36.2.2 client . . . . .	284
13.36.2.3 connect . . . . .	284
13.36.2.4 data . . . . .	284
13.36.2.5 sock . . . . .	284
13.37sockstruct Union Reference . . . . .	284
13.37.1 Detailed Description . . . . .	285
13.37.2 Field Documentation . . . . .	285
13.37.2.1 sa . . . . .	285
13.37.2.2 sa4 . . . . .	285
13.37.2.3 sa6 . . . . .	285
13.37.2.4 ss . . . . .	285
13.37.2.5 un . . . . .	285
13.38ssldata Struct Reference . . . . .	286
13.38.1 Detailed Description . . . . .	286
13.38.2 Field Documentation . . . . .	286
13.38.2.1 bio . . . . .	286
13.38.2.2 ctx . . . . .	286
13.38.2.3 flags . . . . .	286
13.38.2.4 meth . . . . .	287
13.38.2.5 parent . . . . .	287
13.38.2.6 ssl . . . . .	287
13.39thread_pvt Struct Reference . . . . .	287

13.39.1 Detailed Description	287
13.39.2 Field Documentation	288
13.39.2.1 cleanup	288
13.39.2.2 data	288
13.39.2.3 flags	288
13.39.2.4 func	288
13.39.2.5 sighandler	288
13.39.2.6 thr	289
13.40threadcontainer Struct Reference	289
13.40.1 Detailed Description	289
13.40.2 Field Documentation	289
13.40.2.1 list	289
13.40.2.2 manager	289
13.41unixclient_sockthread Struct Reference	289
13.41.1 Detailed Description	290
13.41.2 Field Documentation	290
13.41.2.1 client	290
13.41.2.2 data	290
13.41.2.3 endpoint	290
13.41.2.4 sock	290
13.42unixserv_sockthread Struct Reference	290
13.42.1 Detailed Description	291
13.42.2 Field Documentation	291
13.42.2.1 data	291
13.42.2.2 mask	291
13.42.2.3 protocol	291
13.42.2.4 read	291
13.42.2.5 sock	292
13.42.2.6 sockpath	292
13.43xml_attr Struct Reference	292
13.43.1 Detailed Description	292
13.43.2 Field Documentation	292
13.43.2.1 name	292
13.43.2.2 value	292
13.44xml_node Struct Reference	293
13.44.1 Detailed Description	293
13.44.2 Field Documentation	293
13.44.2.1 attrs	293
13.44.2.2 key	293
13.44.2.3 name	293

13.44.2.4 nodeptr . . . . .	293
13.44.2.5 value . . . . .	294
13.45xml_node_iter Struct Reference . . . . .	294
13.45.1 Detailed Description . . . . .	294
13.45.2 Field Documentation . . . . .	294
13.45.2.1 cnt . . . . .	294
13.45.2.2 curpos . . . . .	294
13.45.2.3 xsearch . . . . .	294
13.46xml_search Struct Reference . . . . .	295
13.46.1 Detailed Description . . . . .	295
13.46.2 Field Documentation . . . . .	295
13.46.2.1 nodes . . . . .	295
13.46.2.2 xmldoc . . . . .	295
13.46.2.3 xpathObj . . . . .	295
13.47xslt_doc Struct Reference . . . . .	296
13.47.1 Detailed Description . . . . .	296
13.47.2 Field Documentation . . . . .	296
13.47.2.1 doc . . . . .	296
13.47.2.2 params . . . . .	296
13.48xslt_param Struct Reference . . . . .	296
13.48.1 Detailed Description . . . . .	296
13.48.2 Field Documentation . . . . .	297
13.48.2.1 name . . . . .	297
13.48.2.2 value . . . . .	297
13.49zobj Struct Reference . . . . .	297
13.49.1 Detailed Description . . . . .	297
13.49.2 Field Documentation . . . . .	297
13.49.2.1 buff . . . . .	297
13.49.2.2 olen . . . . .	298
13.49.2.3 zlen . . . . .	298
<b>14 File Documentation . . . . .</b>	<b>299</b>
14.1 doxygen/dox/buckets.dox File Reference . . . . .	299
14.2 doxygen/dox/examples.dox File Reference . . . . .	299
14.3 doxygen/dox/index.dox File Reference . . . . .	299
14.4 doxygen/dox/main.dox File Reference . . . . .	299
14.5 doxygen/dox/modules.dox File Reference . . . . .	299
14.6 doxygen/dox/refobj.dox File Reference . . . . .	299
14.7 doxygen/dox/sockets.dox File Reference . . . . .	299
14.8 doxygen/dox/sockex.dox File Reference . . . . .	299



14.9 doxygen/dox/thread.dox File Reference . . . . .	299
14.10doxygen/examples/socket.c File Reference . . . . .	299
14.10.1 Detailed Description . . . . .	300
14.10.2 Function Documentation . . . . .	300
14.10.2.1 accept_func . . . . .	300
14.10.2.2 client_func . . . . .	300
14.10.2.3 FRAMEWORK_MAIN . . . . .	301
14.10.2.4 server_func . . . . .	302
14.10.2.5 socktest . . . . .	302
14.10.2.6 unixsocktest . . . . .	303
14.11socket.c . . . . .	304
14.12src/socket.c File Reference . . . . .	306
14.12.1 Detailed Description . . . . .	307
14.13socket.c . . . . .	307
14.14src/config.c File Reference . . . . .	314
14.14.1 Detailed Description . . . . .	315
14.15config.c . . . . .	315
14.16src/curl.c File Reference . . . . .	319
14.16.1 Detailed Description . . . . .	321
14.17curl.c . . . . .	321
14.18src/fileutil.c File Reference . . . . .	326
14.18.1 Detailed Description . . . . .	326
14.19fileutil.c . . . . .	327
14.20src/include/dtsapp.h File Reference . . . . .	328
14.20.1 Detailed Description . . . . .	340
14.21dtsapp.h . . . . .	341
14.22src/interface.c File Reference . . . . .	350
14.22.1 Detailed Description . . . . .	352
14.23interface.c . . . . .	352
14.24src/iputil.c File Reference . . . . .	361
14.24.1 Detailed Description . . . . .	362
14.25iputil.c . . . . .	362
14.26src/libxml2.c File Reference . . . . .	368
14.26.1 Detailed Description . . . . .	369
14.27libxml2.c . . . . .	369
14.28src/libxslt.c File Reference . . . . .	377
14.28.1 Detailed Description . . . . .	378
14.29libxslt.c . . . . .	378
14.30src/lookup3.c File Reference . . . . .	381
14.30.1 Detailed Description . . . . .	382

14.31lookup3.c . . . . .	382
14.32src/main.c File Reference . . . . .	396
14.32.1 Detailed Description . . . . .	396
14.33main.c . . . . .	396
14.34src/nf_ctrack.c File Reference . . . . .	400
14.34.1 Detailed Description . . . . .	400
14.35nf_ctrack.c . . . . .	401
14.36src/nf_queue.c File Reference . . . . .	404
14.36.1 Detailed Description . . . . .	405
14.37nf_queue.c . . . . .	405
14.38src/openldap.c File Reference . . . . .	409
14.38.1 Detailed Description . . . . .	411
14.38.2 Function Documentation . . . . .	411
14.38.2.1 _dtsldapsearch . . . . .	411
14.39openldap.c . . . . .	412
14.40src/radius.c File Reference . . . . .	430
14.40.1 Detailed Description . . . . .	431
14.41radius.c . . . . .	431
14.42src/refobj.c File Reference . . . . .	438
14.42.1 Detailed Description . . . . .	439
14.43refobj.c . . . . .	439
14.44src/rfc6296.c File Reference . . . . .	447
14.44.1 Detailed Description . . . . .	447
14.45rfc6296.c . . . . .	447
14.46src/sslutil.c File Reference . . . . .	449
14.46.1 Detailed Description . . . . .	451
14.47sslutil.c . . . . .	451
14.48src/thread.c File Reference . . . . .	460
14.48.1 Detailed Description . . . . .	461
14.49thread.c . . . . .	461
14.50src/unixsock.c File Reference . . . . .	466
14.50.1 Detailed Description . . . . .	467
14.51unixsock.c . . . . .	467
14.52src/util.c File Reference . . . . .	471
14.52.1 Detailed Description . . . . .	473
14.53util.c . . . . .	473
14.54src/winiface.cpp File Reference . . . . .	477
14.54.1 Detailed Description . . . . .	477
14.55winiface.cpp . . . . .	477
14.56src/zlib.c File Reference . . . . .	479

<b>CONTENTS</b>	<b>xxv</b>
14.56.1 Detailed Description . . . . .	479
14.57zlib.c . . . . .	480
<b>15 Example Documentation</b>	<b>483</b>
15.1 socket.c . . . . .	483
<b>Index</b>	<b>485</b>



# 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](mailto:gregory@distrotech.co.za)>

#### Date

2010-

#### Copyright

GNU Public Licence.



## 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) {
#ifdef __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 simple by nature well defined and logical. Using pointers is almost imperative 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 freed.

So assuming all data structs are pointers the only access to the data is via these pointers the pointers can be copied and overwritten 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 reference it using [objref\(\)](#) and release the reference with [objunref\(\)](#) if the reference count is 0 the object will be freed the current count is returned by [objcnt\(\)](#).

lets look at the code again but using referenced 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(struct 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 if 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 voodoo 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 allocated 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 benefiting 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 respectively, taking into account the availability of memory and the benefits this will be acceptable.

One option is to drop support for [objsize\(\)](#) this will save 4bytes and 8 bytes respectively removing the magic cookie is not recommended.



## 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 following 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 / memory 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 their 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 sequentially is faster than non sequential 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 an element to the list you set the last element in the list's 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 effectively equivalent as to iterating through an array except for the loss of speed not being 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 requiring 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 in fact an array of linked lists see the [bucket\\_list](#) structure.

The list is an array of  $2^{\text{bucketbits}}$  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 efficient 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 based on their hash, this allows the algorithm to search forward or backward theoretically only ever having to traverse 64 elements of 8192.

Using this hybrid approach gives us a good compromise and benefits 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 being faster with faster memory they are 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);
}
```

That's that folks use [addtobucket\(\)](#) to add an 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\(\)](#) respectively.

To implement your own iterator 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 program 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 initialization. what function is called and what data is provided is up to the programmer.

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 caceable.

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, additionally 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 initialization 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 require 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/Writing 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 equivalent 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 = dtls1_init("certs/cacert.pem", "certs/server-cert.pem",
            "certs/server-key.pem", SSL_VERIFY_PEER | SSL_VERIFY_CLIENT_ONCE);
        ssl_c = dtls1_init("certs/cacert.pem", "certs/client-cert.pem",
            "certs/client-key.pem", SSL_VERIFY_NONE);
        ssl_c2 = dtls1_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);
}

#ifdef __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) {
#ifdef __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 \*progrname, 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 . . . . .	36
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 . . . . .	89
Linux network interface functions . . . . .	93
INI Style config file Interface . . . . .	107
Radius client interface . . . . .	114
Micelaneous utilities. . . . .	122
Hashing and digest functions . . . . .	129
MD5 Hashing and digest functions . . . . .	130
SHA1 Hashing and digest functions . . . . .	132
SHA2-256Hashing and digest functions . . . . .	134
SHA2-512 Hashing and digest functions . . . . .	136
IPv4 and IPv6 functions . . . . .	138
IPv4 functions . . . . .	141
IPv6 functions . . . . .	150
File utility functions . . . . .	154
Openldap/SASL Interface . . . . .	157
XML Interface . . . . .	176
XSLT Interface . . . . .	191
CURL Url interface. . . . .	196
Zlib Interface . . . . .	206
Burtle Bob hash algorythim. . . . .	209
IPv6 Nat Mapping . . . . .	225
Windows Support . . . . .	229
Distrotech Application Library (Todo) . . . . .	232
Linux Netfilter . . . . .	233
Connection Tracking . . . . .	234
Queue interface . . . . .	239



## Chapter 10

# Data Structure Index

### 10.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">basic_auth</a>	Basic authentication structure . . . . .	243
<a href="#">blist_obj</a>	Entry in a bucket list . . . . .	244
<a href="#">bucket_list</a>	Bucket list, hold hashed objects in buckets . . . . .	245
<a href="#">bucket_loop</a>	Bucket iterator . . . . .	246
<a href="#">config_category</a>	Configuration file category . . . . .	248
<a href="#">config_entry</a>	Configuration category entry . . . . .	248
<a href="#">config_file</a>	Config file . . . . .	249
<a href="#">curl_post</a>	HTTP post data structure . . . . .	250
<a href="#">curlbuf</a>	Buffer containing the result of a curl transaction . . . . .	251
<a href="#">framework_core</a>	Application framework data . . . . .	252
<a href="#">fwsocket</a>	Socket data structure . . . . .	254
<a href="#">ifinfo</a>	Data structure containing interface information . . . . .	256
<a href="#">ipaddr_req</a>	IP Netlink IP addr request . . . . .	257
<a href="#">iplink_req</a>	IP Netlink request . . . . .	258
<a href="#">ldap_add</a>	LDAP Add structure . . . . .	259
<a href="#">ldap_attr</a>	LDAP attribute . . . . .	260
<a href="#">ldap_attrval</a>	LDAP attribute value . . . . .	261
<a href="#">ldap_conn</a>	LDAP connection . . . . .	262
<a href="#">ldap_entry</a>	LDAP entry . . . . .	264

<a href="#">ldap_modify</a>	LDAP Modify structure . . . . .	265
<a href="#">ldap_modreq</a>	LDAP mod request . . . . .	266
<a href="#">ldap_modval</a>	Linked list of mod values . . . . .	267
<a href="#">ldap_rdn</a>	LDAP Relative distinguished name linked list . . . . .	268
<a href="#">ldap_results</a>	LDAP results . . . . .	269
<a href="#">ldap_simple</a>	LDAP Simple bind . . . . .	270
<a href="#">natmap</a>	RFC6296 Nat map . . . . .	270
<a href="#">nfq_queue</a>	. . . . .	272
<a href="#">nfq_struct</a>	. . . . .	272
<a href="#">pseudohdr</a>	IPv4 header structur to cast a packet too . . . . .	273
<a href="#">radius_connection</a>	Radius connection . . . . .	274
<a href="#">radius_packet</a>	Radius Packet . . . . .	275
<a href="#">radius_server</a>	Radius Server . . . . .	277
<a href="#">radius_session</a>	Radius session . . . . .	279
<a href="#">ref_obj</a>	Internal structure of all referenced objects . . . . .	280
<a href="#">sasl_defaults</a>	SASL Paramaters used in authentication . . . . .	282
<a href="#">socket_handler</a>	Socket handling thread data . . . . .	283
<a href="#">sockstruct</a>	Socket union describing all address types . . . . .	284
<a href="#">ssldata</a>	SSL data structure for enabling encryption on sockets . . . . .	286
<a href="#">thread_pvt</a>	Thread struct used to create threads data needs to be first element . . . . .	287
<a href="#">threadcontainer</a>	Global threads data . . . . .	289
<a href="#">unixclient_sockthread</a>	Unix socket client data structure . . . . .	289
<a href="#">unixserv_sockthread</a>	Unix socket server data structure . . . . .	290
<a href="#">xml_attr</a>	XML attribute name value pair . . . . .	292
<a href="#">xml_node</a>	Reference to a XML Node . . . . .	293
<a href="#">xml_node_iter</a>	Iterator to traverse nodes in a xpath . . . . .	294
<a href="#">xml_search</a>	XML xpath search result . . . . .	295
<a href="#">xslt_doc</a>	XSLT Document . . . . .	296
<a href="#">xslt_param</a>	XSLT Parameter name/value pair . . . . .	296
<a href="#">zobj</a>	Zlib buffer used for compression and decompression . . . . .	297

# Chapter 11

## File Index

### 11.1 File List

Here is a list of all files with brief descriptions:

doxygen/examples/ <a href="#">socket.c</a>	
Echo server using 1 server and 2 clients . . . . .	304
src/ <a href="#">config.c</a>	
INI style config file interface . . . . .	315
src/ <a href="#">curl.c</a>	
CURL Interface . . . . .	321
src/ <a href="#">fileutil.c</a>	
File utilities to test files (fstat) . . . . .	327
src/ <a href="#">interface.c</a>	
Wrapper around Linux libnetlink for managing network interfaces . . . . .	352
src/ <a href="#">iputil.c</a>	
IPv4 And IPv6 Utilities . . . . .	362
src/ <a href="#">libxml2.c</a>	
XML Interface . . . . .	369
src/ <a href="#">libxslt.c</a>	
XSLT Interface . . . . .	378
src/ <a href="#">lookup3.c</a>	
By Bob Jenkins, May 2006, Public Domain . . . . .	382
src/ <a href="#">main.c</a>	
Application framework . . . . .	396
src/ <a href="#">nf_ctrack.c</a>	
Linux Netfilter Connection Tracking . . . . .	401
src/ <a href="#">nf_queue.c</a>	
Linux netfilter queue interface . . . . .	405
src/ <a href="#">openldap.c</a>	
Openldap/SASL Implementation . . . . .	412
src/ <a href="#">radius.c</a>	
Simple radius client implementation . . . . .	431
src/ <a href="#">refobj.c</a>	
Referenced Lockable Objects . . . . .	439
src/ <a href="#">rfc6296.c</a>	
Implementation of RFC6296 . . . . .	447
src/ <a href="#">socket.c</a>	
Allocate and initialise a socket for use as a client or server . . . . .	307
src/ <a href="#">sslutil.c</a>	
TLSv1 SSLv2 SSLv3 DTLSv1 support . . . . .	451
src/ <a href="#">thread.c</a>	
Functions for starting and managing threads . . . . .	461

<a href="#">src/unixsock.c</a>	
Attach a thread to a unix socket start a new thread on connect . . . . .	467
<a href="#">src/util.c</a>	
Utilities commonly used . . . . .	473
<a href="#">src/winiface.cpp</a>	
Various routines for supporting Windows also requires C++ . . . . .	477
<a href="#">src/zlib.c</a>	
Simplified implementation of zlib functions . . . . .	480
<a href="#">src/include/dtsapp.h</a>	
DTS Application library API Include file . . . . .	341

# 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.*
- [Miscellaneous utilities.](#)  
*Utilities commonly used.*
- [IPv4 and IPv6 functions](#)  
*Helper functions for various calculations.*
- [File utility functions](#)  
*Convince wrappers around 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](#)(progrname, name, email, www, year, runfile, flags, sighfunc)  
*A macro to replace main() with initialization 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 \*progrname, 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 initialization and daemonization code.

#### Note

Argument count is argc and arguments is array argv.

#### See Also

[framework\\_flags](#)  
[framework\\_mkcore\(\)](#)  
[framework\\_init\(\)](#)

#### Parameters

<i>progname</i>	Descriptive program name.
<i>name</i>	Copyright holders name.
<i>email</i>	Copyright holders email.
<i>www</i>	Web address.
<i>year</i>	Copyright year.
<i>runfile</i>	Application runfile.
<i>flags</i>	Application flags.
<i>sighfunc</i>	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

<i>argc</i>	Argument count.
<i>argv</i>	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

<i>sig</i>	Signal been handled.
<i>si</i>	Sa sigaction.
<i>unused</i>	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](#).

```
{
FRAMEWORK_FLAG_DAEMON    = 1 << 0,
FRAMEWORK_FLAG_NOGNU     = 1 << 1,
FRAMEWORK_FLAG_DAEMONLOCK = 1 << 2
};
```

### 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\(\)](#).

```

    {
        struct framework_core *ci = framework_core_info;
#ifdef __WIN32__
        pid_t    forkpid;

        /* fork and die daemonize*/
        forkpid = fork();
        if (forkpid > 0) {
            /* im all grown up and can pass onto child*/
            exit(0);
        } else if (forkpid < 0) {
            /* could not fork*/
            exit(-1);
        }

        setsid();

        /* Dont want these as a daemon*/
        signal(SIGTSTP, SIG_IGN);
        signal(SIGCHLD, SIG_IGN);
#endif

        /*delayed lock file from FRAMEWORK_MAIN / framework_init*/
        if (ci && (ci->flags & FRAMEWORK_FLAG_DAEMONLOCK
        )) {
            if ((ci->flock = lockpidfile(ci->runfile)) < 0)
            {
                printf("Could not lock pid file Exiting\n");
                while(framework_core_info) {
                    objunref(framework_core_info);
                }
                exit (-1);
            }
            objunref(ci);
        }
    }
}

```

### 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 jointreads.  
framework information configured by framework\_mkcore will be freed on exit.

## Parameters

<i>argc</i>	Argument count argv[0] will be program name.
<i>argv</i>	Argument array.
<i>callback</i>	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](#).

```

{
    struct framework_core *ci = framework_core_info;
    int ret = 0;

    seedrand();
    sslstartup();

    /*print out a GNU licence summary*/
    if (ci && !(ci->flags & FRAMEWORK_FLAG_NOGNU)) {
        printgnu(ci->progname, ci->year, ci->developer
            , ci->email, ci->www);
    }

    /* grab a ref for framework_core_info to be used latter*/
    if (ci && ci->flags & FRAMEWORK_FLAG_DAEMONLOCK
        ) {
        objref(ci);
    }

    /* fork the process to daemonize it*/
    if (ci && ci->flags & FRAMEWORK_FLAG_DAEMON) {
        daemonize();
    }

    /* write pid to lockfile this should be done post daemonize*/
    if (ci && !(ci->flags & FRAMEWORK_FLAG_DAEMONLOCK
        )) {
        if ((ci->flock = lockpidfile(ci->runfile)) < 0)
        {
            printf("Could not lock pid file Exiting\n");
            return -1;
        }
    }

#ifdef __WIN32__
    /* interrupt handler close clean on term so physical is reset*/
    configure_sigact(framework_core_info->sa);
#endif

    /*run the code from the application*/
    if (callback) {
        ret = callback(argc, argv);
        /* wait for all threads to end*/
        stopthreads(1);
    }

    /* turn off the lights*/
    objunref(ci);
    if (framework_core_info && framework_core_info->flags &
        FRAMEWORK_FLAG_DAEMONLOCK) {
        objunref(framework_core_info);
    }
    unrefconfigfiles();
    return (ret);
}

```

**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 default 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**

<i>progname</i>	Descriptive program name.
<i>name</i>	Copyright holder.
<i>email</i>	Copyright email address.
<i>web</i>	Website address.
<i>year</i>	Copyright year.
<i>runfile</i>	Run file that will store the pid and be locked (flock).
<i>flags</i>	Application flags.
<i>sigfunc</i>	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`.

```

{
    struct framework_core *core_info;
    if (framework_core_info) {
        objunref(framework_core_info);
        framework_core_info = NULL;
    }

    if (!(core_info = objalloc(sizeof(*core_info), framework_free))) {
        return;
    }

#ifdef __WIN32__
    if (core_info && !(core_info->sa = malloc(sizeof(*core_info->sa))) {
        free(core_info);
        return;
    }
#endif

    ALLOC_CONST(core_info->developer, name);
    ALLOC_CONST(core_info->email, email);
    ALLOC_CONST(core_info->www, web);
    ALLOC_CONST(core_info->runfile, runfile);
    ALLOC_CONST(core_info->progname, progname);
    core_info->year = year;
    core_info->flags = flags;
#ifdef __WIN32__
    core_info->sig_handler = sigfunc;
#endif
    /* Pass reference to static system variable*/
    framework_core_info = core_info;
}

```

**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_DAEMON-LOCK`

## Parameters

<i>runfile</i>	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\(\)](#).

```

{
    int lck_fd = 0;
#ifdef __WIN32__
    char pidstr[12];
    pid_t mypid;

    mypid = getpid();
    sprintf(pidstr, "%i\n", (int)mypid);
    if (runfile && ((lck_fd = open(runfile, O_RDWR|O_CREAT, 0640)
        ) > 0) && (!flock(lck_fd, LOCK_EX | LOCK_NB))) {
        if (write(lck_fd, pidstr, strlen(pidstr)) < 0) {
            close(lck_fd);
            lck_fd = -1;
        }
        /* file was opened and not locked*/
    } else if (runfile && lck_fd) {
        close(lck_fd);
        lck_fd = -1;
    }
#endif
    return (lck_fd);
}

```

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

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

Definition at line 78 of file [main.c](#).

Referenced by [framework\\_init\(\)](#).

```

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

```

}

## 12.2 Referenced Lockable Objects

Utilities for managing referenced lockable objects.

### Modules

- [Hashed bucket linked lists of referenced objects](#)  
*Store references in and retrieve 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 0xdead0de`  
*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);\nobj->flags &= ~flag;\nobjunlock(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) {\n    return objalloc(sz, &classtype::dts_unref_classtype);\n}\nvoid operator delete(void *obj) {\n}\nstatic void dts_unref_classtype(void *data) {\n    delete (classtype*)data;\n}\n~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 0xdead0de

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

<i>data</i>	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

<i>size</i>	Size of the data buffer to allocate in addition to the reference.
<i>destructor</i>	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 [\\_dtsldapsearch\(\)](#), [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\(\)](#).

```

{
    struct ref_obj *ref;
    int asize;
    char *robj;

    asize = size + refobj_offset;

    if ((robj = malloc(asize))) {
        memset(robj, 0, asize);
        ref = (struct ref_obj *)robj;
        pthread_mutex_init(&ref->lock, NULL);
        ref->magic = REFOBJ_MAGIC;
        ref->cnt = 1;
        ref->data = robj + refobj_offset;
        ref->size = asize;
        ref->destroy = destructor;
        return (ref->data);
    }
    return NULL;
}

```

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

Return a reference to copy of a buffer.

#### Parameters

<i>orig</i>	Original buffer to copy.
-------------	--------------------------

#### Returns

Reference to new instance of orig.

Definition at line 330 of file [refobj.c](#).

References [objalloc\(\)](#).

```

{
    int len = strlen(orig) + 1;
    void *nobj;

    if ((nobj = objalloc(len, NULL))) {

```

```

        memcpy(nobj, orig, len);
    }
    return nobj;
}

```

#### 12.2.4.3 int objcnt ( void \* data )

Return current reference count.

##### Parameters

<i>data</i>	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\(\)](#).

```

{
    char *ptr = data;
    int ret = -1;
    struct ref_obj *ref;

    if (!data) {
        return (ret);
    }

    ptr = ptr - refobj_offset;
    ref = (struct ref_obj *)ptr;

    if (ref->magic == REFOBJ_MAGIC) {
        pthread_mutex_lock(&ref->lock);
        ret = ref->cnt;
        pthread_mutex_unlock(&ref->lock);
    }
    return (ret);
}

```

#### 12.2.4.4 int objlock ( void \* data )

Lock the reference.

##### Parameters

<i>data</i>	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 [\\_dtsldapsearch\(\)](#), [accept\\_socket\(\)](#), [addtobucket\(\)](#), [bucket\\_list\\_cnt\(\)](#), [create\\_kernmac\(\)](#), [create\\_kernvlan\(\)](#), [curl\\_postitem\(\)](#), [curlinit\(\)](#), [dtls\\_listenssl\(\)](#), [dtlshandltimeout\(\)](#), [dtlstimeout\(\)](#), [dtls\\_serveropts\(\)](#), [framework\\_mkthread\(\)](#), [get\\_iface\\_index\(\)](#), [jointhreads\(\)](#), [ldap\\_doaddd\(\)](#), [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\(\)](#).

```

    {
        char *ptr = data;
        struct ref_obj *ref;

        ptr = ptr - refobj_offset;
        ref = (struct ref_obj *)ptr;

        if (data && ref->magic == REFOBJ_MAGIC) {
            pthread_mutex_lock(&ref->lock);
        }
        return (0);
    }

```

#### 12.2.4.5 int objref ( void \* data )

Reference a object.

##### Parameters

<i>data</i>	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 [\\_dtsldapsearch\(\)](#), [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\\_simplebind\(\)](#), [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\(\)](#).

```

    {
        char *ptr = data;
        struct ref_obj *ref;
        int ret = 0;

        ptr = ptr - refobj_offset;
        ref = (struct ref_obj *)ptr;

        if (!data || !ref || (ref->magic != REFOBJ_MAGIC)) {
            return (ret);
        }

        /*double check just incase im gone*/
        if (!pthread_mutex_lock(&ref->lock)) {
            if ((ref->magic == REFOBJ_MAGIC) && (ref->cnt > 0))
            {
                ref->cnt++;
                ret = ref->cnt;
            }
            pthread_mutex_unlock(&ref->lock);
        }

        return (ret);
    }

```

#### 12.2.4.6 int objsize ( void \* data )

Size requested for data.

##### Note

the size of the data is returned.

##### Parameters

<i>data</i>	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](#).

```

{
    char *ptr = data;
    int ret = 0;
    struct ref_obj *ref;

    if (!data) {
        return (ret);
    }

    ptr = ptr - refobj_offset;
    ref = (struct ref_obj *)ptr;

    if (ref->magic == REFOBJ_MAGIC) {
        pthread_mutex_lock(&ref->lock);
        ret = ref->size - refobj_offset;
        pthread_mutex_unlock(&ref->lock);
    }
    return (ret);
}

```

#### 12.2.4.7 int objtrylock ( void \* data )

Try lock a reference.

##### Parameters

<i>data</i>	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](#).

```

{
    char *ptr = data;
    struct ref_obj *ref;

    ptr = ptr - refobj_offset;
    ref = (struct ref_obj *)ptr;

    if (ref->magic == REFOBJ_MAGIC) {
        return ((pthread_mutex_trylock(&ref->lock)) ? -1 : 0);
    }
    return (-1);
}

```

12.2.4.8 `int objunlock ( void * data )`

Unlock a reference.

## Parameters

<i>data</i>	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 [\\_dtsldapsearch\(\)](#), [accept\\_socket\(\)](#), [addtobucket\(\)](#), [bucket\\_list\\_cnt\(\)](#), [create\\_kernmac\(\)](#), [create\\_kernvlan\(\)](#), [curl\\_postitem\(\)](#), [curlinit\(\)](#), [dtls\\_listenssl\(\)](#), [dtlshandltimeout\(\)](#), [dtltimeout\(\)](#), [dtls\\_serveropts\(\)](#), [framework\\_mkthread\(\)](#), [get\\_iface\\_index\(\)](#), [jointhreads\(\)](#), [ldap\\_doaddd\(\)](#), [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\(\)](#).

```

{
    char *ptr = data;
    struct ref_obj *ref;

    ptr = ptr - refobj_offset;
    ref = (struct ref_obj *)ptr;

    if (ref->magic == REFOBJ_MAGIC) {
        pthread_mutex_unlock(&ref->lock);
    }
    return (0);
}

```

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

<i>data</i>	Data we are dropping a reference for
-------------	--------------------------------------

## Returns

-1 on error or the reference 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 [\\_dtsldapsearch\(\)](#), [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\_doaddd(), 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().

```

{
    char *ptr = data;
    struct ref_obj *ref;
    int ret = -1;

    if (!data) {
        return (ret);
    }

    ptr = ptr - refobj_offset;
    ref = (struct ref_obj *)ptr;

    if ((ref->magic == REFOBJ_MAGIC) && (ref->cnt)) {
        pthread_mutex_lock(&ref->lock);
        ref->cnt--;
        ret = ref->cnt;
        /* free the object its no longer in use*/
        if (!ret) {
            ref->magic = 0;
            ref->size = 0;
            ref->data = NULL;
            if (ref->destroy) {
                ref->destroy(data);
            }
            pthread_mutex_unlock(&ref->lock);
            pthread_mutex_destroy(&ref->lock);
            free(ref);
        } else {
            pthread_mutex_unlock(&ref->lock);
        }
    }
    return (ret);
}

```



## 12.3 Hashed bucket linked lists of referenced objects

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

#### See Also

[Hashed Bucket Lists](#)  
[Burtle Bob hash algorithm.](#)

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

<i>bitmask</i>	Number of buckets to create $2^{\wedge}$ bitmask.
<i>hash_function</i>	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

<i>data</i>	Reference held by the list.
<i>data2</i>	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

<i>data</i>	Data or key to calculate hash from.
<i>key</i>	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

<i>blist</i>	Bucket list to add too.
<i>data</i>	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 [\\_dtsldapsearch\(\)](#), [add\\_radserver\(\)](#), [framework\\_mkthread\(\)](#), [process\\_config\(\)](#), [rfc6296\\_map\\_add\(\)](#), and [xslt\\_addparam\(\)](#).

```

char *ptr = data;
struct ref_obj *ref;
struct blist_obj *lhead, *tmp;
unsigned int hash, bucket;

if (!objref(blist)) {
    return (0);
}

if (!objref(data)) {
    objunref(blist);
    return (0);
}

ptr = ptr - refobj_offset;
ref = (struct ref_obj *)ptr;

hash = gethash(blist, data, 0);
bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
    bucketbits) - 1));

pthread_mutex_lock(&blist->locks[bucket]);
lhead = blist->list[bucket];
/*no head or non null head*/
if (!lhead || lhead->prev) {
    if (!(tmp = malloc(sizeof(*tmp)))) {
        {

```

```

        pthread_mutex_unlock(&blist->locks[bucket]);
        objunref(data);
        objunref(blist);
        return (0);
    }
    memset(tmp, 0, sizeof(*tmp));
    tmp->hash = hash;
    tmp->data = ref;

    /*there is no head*/
    if (!lhead) {
        blist->list[bucket] = tmp;
        tmp->prev = tmp;
        tmp->next = NULL;
        /*become new head*/
    } else if (hash < lhead->hash) {
        tmp->next = lhead;
        tmp->prev = lhead->prev;
        lhead->prev = tmp;
        blist->list[bucket] = tmp;
        /*new tail*/
    } else if (hash > lhead->prev->hash) {
        tmp->prev = lhead->prev;
        tmp->next = NULL;
        lhead->prev->next = tmp;
        lhead->prev = tmp;
        /*insert entry*/
    } else {
        lhead = blist_gotohash(lhead, hash, blist->bucketbits);
        tmp->next = lhead->next;
        tmp->prev = lhead;

        if (lhead->next) {
            lhead->next->prev = tmp;
        } else {
            blist->list[bucket]->prev = tmp;
        }
        lhead->next = tmp;
    }
} else {
    /*set NULL head*/
    lhead->data = ref;
    lhead->prev = lhead;
    lhead->next = NULL;
    lhead->hash = hash;
}

blist->version[bucket]++;
pthread_mutex_unlock(&blist->locks[bucket]);

objlock(blist);
blist->count++;
objunlock(blist);
objunref(blist);

return (1);
}

```

### 12.3.3.2 int bucket\_list\_cnt ( struct bucket\_list \* blist )

Return number of items in the list.

#### Parameters

<i>blist</i>	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\(\)](#).

{

```

int ret = -1;

if (blist) {
    objlock(blist);
    ret = blist->count;
    objunlock(blist);
}
return (ret);
}

```

### 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

<i>blist</i>	Bucket list to search.
<i>key</i>	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\(\)](#).

```

{
    struct blist_obj *entry;
    int hash, bucket;

    if (!blist) {
        return (NULL);
    }

    hash = gethash(blist, key, 1);
    bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
        bucketbits) - 1));

    pthread_mutex_lock(&blist->locks[bucket]);
    entry = blist_gotohash(blist->list[bucket], hash + 1, blist->bucketbits
    );
    if (entry && entry->data) {
        objref(entry->data->data);
    } else
        if (!entry) {
            pthread_mutex_unlock(&blist->locks[bucket]);
            return NULL;
        }

    pthread_mutex_unlock(&blist->locks[bucket]);

    if (entry->data && (entry->hash == hash)) {
        return (entry->data->data);
    } else
        if (entry->data) {
            objunref(entry->data->data);
        }

    return NULL;
}

```

#### 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

<i>blist</i>	Bucket list to iterate through.
<i>callback</i>	Callback to call for each iteration.
<i>data2</i>	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\(\)](#).

```

    {
        struct bucket_loop *bloop;
        void *data;

        if (!blist || !callback) {
            return;
        }

        bloop = init_bucket_loop(blist);
        while(blist && bloop && (data = next_bucket_loop(bloop))) {
            callback(data, data2);
            objunref(data);
        }
        objunref(bloop);
    }

```

#### 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 [\\_dtsldapsearch\(\)](#), [add\\_radserver\(\)](#), [ldap\\_addinit\(\)](#), [ldap\\_modifyinit\(\)](#), [rfc6296\\_map\\_add\(\)](#), [socket-server\(\)](#), [startthreads\(\)](#), [xslt\\_clearparam\(\)](#), and [xslt\\_open\(\)](#).

```

    {
        struct bucket_list *new;
        short int buckets, cnt;

        buckets = (1 << bitmask);

        /* allocate session bucket list memory size of the struct plus a list lock
           and version for each bucket*/
        if (!(new = objalloc(sizeof(*new) + (sizeof(void *) + sizeof(
            pthread_mutex_t) + sizeof(size_t)) * buckets, empty_buckets))) {
            return NULL;
        }

        /*initialise each bucket*/
        new->bucketbits = bitmask;
        new->list = (void *)((char *)new + sizeof(*new));
        for (cnt = 0; cnt < buckets; cnt++) {
            if ((new->list[cnt] = malloc(sizeof(*new->list[cnt]))) {
                memset(new->list[cnt], 0, sizeof(*new->list[cnt]));
            }
        }

        /*next pointer is pointer to locks*/
        new->locks = (void *)&new->list[buckets];
    }

```

```

    for (cnt = 0; cnt < buckets; cnt++) {
        pthread_mutex_init(&new->locks[cnt], NULL);
    }

    /*Next up version array*/
    new->version = (void *)&new->locks[buckets];

    new->hash_func = hash_function;

    return (new);
}

```

### 12.3.3.6 struct bucket\_loop\* init\_bucket\_loop ( struct bucket\_list \* blist ) [read]

Create a bucket list iterator to safely iterate the list.

#### Parameters

<i>blist</i>	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\\_doaddd\(\)](#), and [ldap\\_domodify\(\)](#).

```

{
    struct bucket_loop *bloop = NULL;

    if (blist && (bloop = objalloc(sizeof(*bloop), free_bloop))) {
        objref(blist);
        bloop->blist = blist;
        bloop->bucket = 0;
        pthread_mutex_lock(&blist->locks[bloop->bucket]);
        bloop->head = blist->list[0];
        if (bloop->head) {
            bloop->head_hash = bloop->head->hash;
        };
        bloop->version = blist->version[0];
        pthread_mutex_unlock(&blist->locks[bloop->bucket]);
    }

    return (bloop);
}

```

### 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

<i>bloop</i>	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\\_doaddd\(\)](#), and [ldap\\_domodify\(\)](#).

```

{
    struct bucket_list *blist = bloop->blist;
    struct ref_obj *entry = NULL;
    void *data = NULL;

    pthread_mutex_lock(&blist->locks[bloop->bucket]);
    if (bloop->head_hash && (blist->version[bloop->bucket]
    ] != bloop->version)) {
        /* bucket has changed unexpectedly i need to ff/rew to hash*/
        bloop->head = blist_gotohash(blist->list[bloop->bucket],
        bloop->head_hash + 1, blist->bucketbits);
        /*if head has gone find next suitable ignore any added*/
        while (bloop->head && (bloop->head->hash < bloop->head_hash
        )) {
            bloop->head = bloop->head->next;
        }
    }

    while (!bloop->head || !bloop->head->prev) {
        pthread_mutex_unlock(&blist->locks[bloop->bucket]);
        bloop->bucket++;
        if (bloop->bucket < (1 << blist->bucketbits)) {
            pthread_mutex_lock(&blist->locks[bloop->bucket]);
            bloop->head = blist->list[bloop->bucket];
        } else {
            return NULL;
        }
    }

    if (bloop->head) {
        bloop->cur = bloop->head;
        entry = (bloop->head->data) ? bloop->head->data : NULL;
        data = (entry) ? entry->data : NULL;
        objref(data);
        bloop->head = bloop->head->next;
        bloop->head_hash = (bloop->head) ? bloop->head->hash
        : 0;
        bloop->cur_hash = (bloop->cur) ? bloop->cur->hash : 0
    }
    pthread_mutex_unlock(&blist->locks[bloop->bucket]);

    return (data);
}

```

#### 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

<i>blist</i>	Bucket list to remove item from.
--------------	----------------------------------

##### See Also

[remove\\_bucket\\_loop](#)

##### Parameters

<i>data</i>	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\(\)](#).

```

{
    struct blist_obj *entry;
    int hash, bucket;

    hash = gethash(blist, data, 0);
    bucket = ((hash >> (32 - blist->bucketbits)) & ((1 << blist->
        bucketbits) - 1));

    pthread_mutex_lock(&blist->locks[bucket]);
    entry = blist_gotohash(blist->list[bucket], hash + 1, blist->bucketbits
    );
    if (entry && entry->hash == hash) {
        if (entry->next && (entry == blist->list[bucket])) {
            entry->next->prev = entry->prev;
            blist->list[bucket] = entry->next;
        } else if (entry->next) {
            entry->next->prev = entry->prev;
            entry->prev->next = entry->next;
        } else if (entry == blist->list[bucket]) {
            blist->list[bucket] = NULL;
        } else {
            entry->prev->next = NULL;
            blist->list[bucket]->prev = entry->prev;
        }
        objunref(entry->data->data);
        free(entry);
        objlock(blist);
        blist->count--;
        blist->version[bucket]++;
        objunlock(blist);
    }
    pthread_mutex_unlock(&blist->locks[bucket]);
}

```

#### 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

Removing 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

<i>bloop</i>	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](#).

```

{
    struct bucket_list *blist = bloop->blist;
    int bucket = bloop->bucket;

    pthread_mutex_lock(&blist->locks[bloop->bucket]);
    /*if the bucket has altered need to verify i can remove*/
    if (bloop->cur_hash && (!bloop->cur || (blist->version[
        bloop->bucket] != bloop->version))) {
        bloop->cur = blist_gotohash(blist->list[bloop->bucket],
        bloop->cur_hash + 1, blist->bucketbits);
        if (!bloop->cur || (bloop->cur->hash != bloop->cur_hash
        )) {
            pthread_mutex_unlock(&blist->locks[bucket]);
            return;
        }
    }
}

```

```
if (!bloop->cur) {
    pthread_mutex_unlock(&blist->locks[bucket]);
    return;
}

if (bloop->cur->next && (bloop->cur == blist->list[bucket]))
{
    bloop->cur->next->prev = bloop->cur->prev;
    blist->list[bucket] = bloop->cur->next;
} else if (bloop->cur->next) {
    bloop->cur->next->prev = bloop->cur->prev;
    bloop->cur->prev->next = bloop->cur->next;
} else if (bloop->cur == blist->list[bucket]) {
    blist->list[bucket] = NULL;
} else {
    bloop->cur->prev->next = NULL;
    blist->list[bucket]->prev = bloop->cur->prev;
}

objunref(bloop->cur->data->data);
free(bloop->cur);
bloop->cur_hash = 0;
bloop->cur = NULL;
blist->version[bucket]++;
bloop->version++;
pthread_mutex_unlock(&blist->locks[bucket]);

objlock(blist);
blist->count--;
objunlock(blist);
}
```

## 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 }
- 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 }
- *Options supplied to framework\_mkthread all defaults are unset.*
- *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

<i>data</i>	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

<i>data</i>	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

<i>data</i>	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 recommended and can lead to memory leaks.

**THREAD\_OPTION\_JOINABLE** Create 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](#).

```

    {
        THREAD_OPTION_CANCEL    = 1 << 0,
        THREAD_OPTION_JOINABLE  = 1 << 1,
        THREAD_OPTION_RETURN    = 1 << 2,
    };

```

#### 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](#).

```

    {
        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
    };

```

## 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 )` [[read](#)]

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

<i>func</i>	Function to run thread on.
<i>cleanup</i>	Cleanup function to run.
<i>sig_handler</i>	Thread signal handler.
<i>data</i>	Data to pass to callbacks.
<i>flags</i>	Options of <a href="#">thread_option_flags</a> 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\(\)](#).

```

{
    struct thread_pvt *thread;
    struct threadcontainer *tc = NULL;

    /*Grab a reference for threads in this scope start up if we can*/
    if (!(tc = (objref(threads)) ? threads : NULL)) {
        if (!thread_can_start) {
            return NULL;
        } else if (!startthreads()) {
            return NULL;
        }
    }
    if (!(tc = (objref(threads)) ? threads : NULL)) {
        return NULL;
    }
}

```

```

}

objlock(tc);
/* dont allow threads if no manager or it not started*/
if ((!tc->manager || !func) && (func != managethread)) {
    /*im shutting down*/
    objunlock(tc);
    objunref(tc);
    return NULL;
} else if (!(thread = objalloc(sizeof(*thread), free_thread))) {
    /* could not create*/
    objunlock(tc);
    objunref(tc);
    return NULL;
}

thread->data = (objref(data)) ? data : NULL;
thread->flags = flags << 16;
thread->cleanup = cleanup;
thread->sighandler = sig_handler;
thread->func = func;
objunlock(tc);

/* start thread and check it*/
if (pthread_create(&thread->thr, NULL, threadwrap, thread) ||
    pthread_kill(thread->thr, 0)) {
    objunref(thread);
    objunref(tc);
    return NULL;
}

/*Activate the thread it needs to be flagged to run or it will die*/
objlock(tc);
addtobucket(tc->list, thread);
setflag(thread, TL_THREAD_RUN);
objunlock(tc);
objunref(tc);

if (testflag(thread, TL_THREAD_RETURN)) {
    return thread;
} else {
    objunref(thread);
    return NULL;
}
}

```

#### 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](#).

```

{
    struct thread_pvt *thr;
    int ret;

    thr = get_thread_from_id();
    ret = (thr) ? testflag(thr, TL_THREAD_RUN) : 0;
    objunref(thr);

    return ret;
}

```

#### 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](#).

```

    {
        struct threadcontainer *tc;

        tc = (objref(threads)) ? threads : NULL;
        if (!tc) {
            return;
        }

        objlock(tc);
        if (tc->manager) {
            setflag(tc->manager, TL_THREAD_JOIN);
            objunlock(tc);
            pthread_join(tc->manager->thr, NULL);
        } else {
            objunlock(tc);
        }
        objunref(tc);
    }

```

#### 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\(\)](#).

```

    {
        struct threadcontainer *tc;

        tc = (objref(threads)) ? threads : NULL;

        if (tc) {
            objunref(tc);
            return 1;
        }

        if (!(tc = objalloc(sizeof(*threads), close_threads))) {
            return 0;
        }

        if (!tc->list && !(tc->list = create_bucketlist(4,
            hash_thread))) {
            objunref(tc);
            return 0;
        }

        threads = tc;
        if (!(tc->manager = framework_mkthread(
            managethread, manage_clean, manager_sig, NULL,
            THREAD_OPTION_JOINABLE |
            THREAD_OPTION_RETURN))) {
            objunref(tc);
            return 0;
        }

        return 1;
    }

```



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

<i>join</i>	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\(\)](#).

```

    {
        struct threadcontainer *tc;

        tc = (objref(threads)) ? threads : NULL;
        if (!tc) {
            return;
        }

        objlock(tc);
        if (tc->manager) {
            setflag(tc->manager, TL_THREAD_STOP);
            if (join) {
                setflag(tc->manager, TL_THREAD_JOIN);
                objunlock(tc);
                pthread_join(tc->manager->thr, NULL);
            } else {
                objunlock(tc);
            }
        } else {
            objunlock(tc);
        }
        objunref(tc);
    }

```

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

<i>sig</i>	Signal to pass.
------------	-----------------

## Returns

1 on success -1 on error.

Definition at line 496 of file [thread.c](#).

References [objunref\(\)](#).

```

    {
        int ret = 0;
#ifdef __WIN32
        struct thread_pvt *thread = NULL;

        if (!(thread = get_thread_from_id())) {
            return 0;
        }

        switch(sig) {
            case SIGUSR1:
            case SIGUSR2:
            case SIGHUP:
            case SIGALRM:
                ret = handle_thread_signal(thread, sig);
                break;
            case SIGINT:
            case SIGTERM:
                ret = handle_thread_signal(thread, sig);
        }
        objunref(thread);
#ifdef __WIN32
        return ret;
    }

```

## 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 controlling 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](#) \* [socketconnect](#) (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

<i>sock</i>	Socket structure data arrived on.
<i>data</i>	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 controlling 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](#).

```
{
    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
};
```

### 12.5.4 Function Documentation

#### 12.5.4.1 struct fwsocket\* accept\_socket ( struct fwsocket \* sock ) [read]

Create and return a socket structure from accept()

Parameters

<i>sock</i>	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](#).

```
{
    struct fwsocket *si;
    socklen_t salen = sizeof(si->addr);

    if (!(si = objalloc(sizeof(*si), clean_fwsocket))) {
        return NULL;
    }

    objlock(sock);
    if ((si->sock = accept(sock->sock, &si->addr.sa, &salen)) < 0)
    {
        objunlock(sock);
        objunref(si);
        return NULL;
    }

    si->type = sock->type;
    si->proto = sock->proto;

    if (sock->ssl) {
        tlsaccept(si, sock->ssl);
    }
    objunlock(sock);
}
```

```

    return (si);
}

```

#### 12.5.4.2 void close\_socket ( struct fwsocket \* sock )

Mark the socket for closure and release the reference.

##### Parameters

<i>sock</i>	Socket to close.
-------------	------------------

Definition at line 79 of file [socket.c](#).

References [objunref\(\)](#), [setflag](#), and [SOCK\\_FLAG\\_CLOSE](#).

Referenced by [sockettest\(\)](#), and [unixsockettest\(\)](#).

```

{
    if (sock) {
        setflag(sock, SOCK_FLAG_CLOSE);
        objunref(sock);
    }
}

```

#### 12.5.4.3 struct fwsocket\* make\_socket ( int family, int type, int proto, void \* ssl ) [read]

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

<i>family</i>	Protocol family.
<i>type</i>	Socket type.
<i>proto</i>	Protocol to be used.
<i>ssl</i>	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\(\)](#).

```

{
    struct fwsocket *si;

    if (!(si = objalloc(sizeof(*si), clean_fwsocket))) {
        return NULL;
    }

    if ((si->sock = socket(family, type, proto)) < 0) {
        objunref(si);
        return NULL;
    }
}

```

```

};

if (ssl) {
    si->ssl = ssl;
}
si->type = type;
si->proto = proto;

return (si);
}

```

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

Return the ip address of a sockstruct addr.

##### Parameters

<i>addr</i>	Socketstruct to return the address for.
<i>buff</i>	Buffer the IP will be copied too.
<i>blen</i>	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](#).

```

{
    if (!buff) {
        return NULL;
    }

    switch (addr->ss.ss_family) {
        case PF_INET:
            inet_ntop(PF_INET, &addr->sa4.sin_addr, buff, blen);
            break;
        case PF_INET6:
            inet_ntop(PF_INET6, &addr->sa6.sin6_addr, buff, blen);
            break;
    }
    return buff;
}

```

#### 12.5.4.5 `struct fwsocket* sockbind ( int family, int stype, int proto, const char * ipaddr, const char * port, void * ssl, int backlog )` [read]

Generic server socket.

##### See Also

[udpbind](#)  
[tcpbind](#)

##### Parameters

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

**Returns**

Reference to socket structure.

Definition at line 290 of file [socket.c](#).

```

return(_opensocket(family, stype, proto, ipaddr, port, ssl, 1,
    backlog));
}

```

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

Generic client socket.

**See Also**

[udpconnect](#)  
[tcpconnect](#)

**Parameters**

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

**Returns**

Reference to socket structure.

Definition at line 250 of file [socket.c](#).

```

return(_opensocket(family, stype, proto, ipaddr, port, ssl, 0, 0));
}

```

**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**

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



Definition at line 493 of file [socket.c](#).

References [startsslclient\(\)](#).

Referenced by [socktest\(\)](#).

```

    {
        startsslclient(sock);
        _start_socket_handler(sock, read, NULL, cleanup, data);
    }

```

#### 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

<i>sock</i>	Socket structure to read from.
<i>buf</i>	Buffer to fill.
<i>num</i>	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\(\)](#).

```

    {
        return (socketread_d(sock, buf, num, NULL));
    }

```

#### 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

<i>sock</i>	Socket structure to read from.
<i>buf</i>	Buffer to fill.
<i>num</i>	Size of the buffer.
<i>addr</i>	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()`.

```

    {
        struct ssldata *ssl = sock->ssl;
        socklen_t salen = sizeof(*addr);
        int ret, err, syserr;

        if (!ssl && !testflag(sock, SOCK_FLAG_SSL)) {
            objlock(sock);
            if (addr && (sock->type == SOCK_DGRAM)) {
                ret = recvfrom(sock->sock, buf, num, 0, &addr->sa, &salen);
            } else {
#ifdef __WIN32
                ret = read(sock->sock, buf, num);
#else
                ret = recv(sock->sock, buf, num, 0);
#endif
            }
            if (ret == 0) {
                sock->flags |= SOCK_FLAG_CLOSE;
            }
            objunlock(sock);
            return (ret);
        } else if (!ssl) {
            return -1;
        }

        objlock(ssl);
        /* ive been shutdown*/
        if (!ssl->ssl) {
            objunlock(ssl);
            return (-1);
        }
        ret = SSL_read(ssl->ssl, buf, num);
        err = SSL_get_error(ssl->ssl, ret);
        if (ret == 0) {
            sock->flags |= SOCK_FLAG_CLOSE;
        }
        objunlock(ssl);
        switch (err) {
            case SSL_ERROR_NONE:
                break;
            case SSL_ERROR_WANT_X509_LOOKUP:
                printf("Want X509\n");
                break;
            case SSL_ERROR_WANT_READ:
                printf("Read Want Read\n");
                break;
            case SSL_ERROR_WANT_WRITE:
                printf("Read Want write\n");
                break;
            case SSL_ERROR_ZERO_RETURN:
            case SSL_ERROR_SSL:
                objlock(sock);
                objunref(sock->ssl);
                sock->ssl = NULL;
                objunlock(sock);
                break;
            case SSL_ERROR_SYSCALL:
                syserr = ERR_get_error();
                if (syserr || (!syserr && (ret == -1))) {
                    printf("R syscall %i %i\n", syserr, ret);
                }
                break;
        }
    }

```

```

        default
        :
        {
            printf("other\n");
            break;
        }
    }
    return (ret);
}

```

**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

<i>sock</i>	Reference to a bound socket.
<i>read</i>	Callback to handle data when ready to read.
<i>acceptfunc</i>	Function to call on connection accept.
<i>cleanup</i>	Thread cleanup function for when the socket closes.
<i>data</i>	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 [sockettest\(\)](#).

```

{
    objlock(sock);
    if (sock->flags & SOCK_FLAG_BIND) {
        if (sock->ssl || !(sock->type == SOCK_DGRAM)) {
            sock->children = create_bucketlist(6,
            hash_socket);
        }
        if (sock->ssl && (sock->type == SOCK_DGRAM)) {
            objunlock(sock);
            dtsl_serveropts(sock);
        } else {
            objunlock(sock);
        }
    } else {
        objunlock(sock);
    }
    _start_socket_handler(sock, read, acceptfunc, cleanup, data);
}

```

**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**

<i>sock</i>	Socket structure to send data too.
<i>buf</i>	Buffer to send.
<i>num</i>	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()`.

```

    }
    return (socketwrite_d(sock, buf, num, NULL));
}

```

**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**

<i>sock</i>	Socket structure to send data too.
<i>buf</i>	Buffer to send.
<i>num</i>	Lengthe of the buffer.
<i>addr</i>	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()`.

```

    {
        struct ssldata *ssl = (sock) ? sock->ssl : NULL;
        int ret, err, syserr;

        if (!sock) {
            return (-1);
        }

        if (!ssl && !testflag(sock, SOCK_FLAG_SSL)) {
            objlock(sock);
            if (addr && (sock->type == SOCK_DGRAM)) {
#ifdef __WIN32
                if (sock->flags & SOCK_FLAG_UNIX) {
                    ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, (const
struct sockaddr *)&addr->un, sizeof(addr->un));
                } else if (sock->flags & SOCK_FLAG_MCAST) {
                    ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &sock->
addr.sa, sizeof(sock->addr.ss));
                } else {
                    ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &addr->sa
, sizeof(*addr));
                }
            }
#else
                if (sock->flags & SOCK_FLAG_MCAST) {
                    ret = sendto(sock->sock, buf, num, 0, &sock->addr.sa,
sizeof(sock->addr.ss));
                } else {
                    ret = sendto(sock->sock, buf, num, 0, &addr->sa, sizeof(*
addr));
                }
            }
#endif
        } else {
#ifdef __WIN32
            if (sock->flags & SOCK_FLAG_MCAST) {
                ret = sendto(sock->sock, buf, num, MSG_NOSIGNAL, &sock->
addr.sa, sizeof(sock->addr.ss));
            } else {
                ret = send(sock->sock, buf, num, MSG_NOSIGNAL);
            }
        }
#else
            if (sock->flags & SOCK_FLAG_MCAST) {
                ret = sendto(sock->sock, buf, num, 0, &sock->addr.sa,
sizeof(sock->addr.ss));
            } else {
                ret = send(sock->sock, buf, num, 0);
            }
        }
#endif
        if (ret == -1) {
            switch(errno) {
                case EBADF:
                case EPIPE:
#ifdef __WIN32
                case ENOTCONN:
                case ENOTSOCK:
#endif
                sock->flags |= SOCK_FLAG_CLOSE;
                break;
            }
        }
        objunlock(sock);
        return (ret);
    } else if (!ssl) {
        return -1;
    }

    if (ssl && ssl->ssl) {
        objlock(ssl);
        if (SSL_state(ssl->ssl) != SSL_ST_OK) {
            objunlock(ssl);
            return (SSL_ERROR_SSL);
        }
        ret = SSL_write(ssl->ssl, buf, num);
        err = SSL_get_error(ssl->ssl, ret);
        objunlock(ssl);
    } else {
        return -1;
    }

    if (ret == -1) {
        setflag(sock, SOCK_FLAG_CLOSE);
    }

    switch(err) {
        case SSL_ERROR_NONE:
            break;
    }

```

```

    case SSL_ERROR_WANT_READ:
        printf("Send Want Read\n");
        break;
    case SSL_ERROR_WANT_WRITE:
        printf("Send Want write\n");
        break;
    case SSL_ERROR_WANT_X509_LOOKUP:
        printf("Want X509\n");
        break;
    case SSL_ERROR_ZERO_RETURN:
    case SSL_ERROR_SSL:
        objlock(sock);
        objunref(sock->ssl);
        sock->ssl = NULL;
        objunlock(sock);
        break;
    case SSL_ERROR_SYSCALL:
        syserr = ERR_get_error();
        if (syserr || (!syserr && (ret == -1))) {
            printf("W syscall %i %i\n", syserr, ret);
        }
        break;
    default:
        printf("other\n");
        break;
}

return (ret);
}

```

#### 12.5.4.13 struct fwsocket\* tcpbind ( const char \* *ipaddr*, const char \* *port*, void \* *ssl*, int *backlog* ) [read]

Generic server socket.

See Also

[udpbind](#)  
[sockbind](#)

##### Parameters

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

##### Returns

Reference to socket structure.

Definition at line 315 of file [socket.c](#).

Referenced by [socktest\(\)](#).

```

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

```

#### 12.5.4.14 struct fwsocket\* tcpconnect ( const char \* *ipaddr*, const char \* *port*, void \* *ssl* ) [read]

TCP Socket client.

## See Also

[sockconnect](#)  
[udpconnect](#)

## Parameters

<i>ipaddr</i>	Ipaddr to connect too.
<i>port</i>	Port to connect too.
<i>ssl</i>	SSL structure to associate with socket.

## Returns

Reference to socket structure.

Definition at line 274 of file [socket.c](#).

Referenced by [socktest\(\)](#).

```

    {
        return (_opensocket(PF_UNSPEC, SOCK_STREAM, IPPROTO_TCP, ipaddr, port, ssl
            , 0, 0));
    }

```

**12.5.4.15** `struct fwsocket* udpbind ( const char * ipaddr, const char * port, void * ssl )` [read]

UDP server socket.

## See Also

[sockbind](#)  
[tcpbind](#)

## Parameters

<i>ipaddr</i>	Ipaddr to connect too.
<i>port</i>	Port to connect too.
<i>ssl</i>	SSL structure to associate with socket.

## Returns

Reference to socket structure.

Definition at line 302 of file [socket.c](#).

Referenced by [socktest\(\)](#).

```

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

```

**12.5.4.16** `struct fwsocket* udpconnect ( const char * ipaddr, const char * port, void * ssl )` [read]

UDP Socket client.

**See Also**

[sockconnect](#)  
[tcpconnect](#)

**Parameters**

<i>ipaddr</i>	Ipaddr to connect too.
<i>port</i>	Port to connect too.
<i>ssl</i>	SSL structure to associate with socket.

**Returns**

Reference to socket structure.

Definition at line [262](#) of file [socket.c](#).

Referenced by [socktest\(\)](#).

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



## 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 declaration 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 [tlaccept](#) (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 [dtls\\_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 to default 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](#) reference will be created and passed on socket initialization.

This is part of the socket interface to support encrypted sockets a [ssldata](#) reference 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 declaration 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](#).

```
{
    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
};
```

## 12.6.5 Function Documentation

### 12.6.5.1 struct fwsocket\* dtls\_listenssl ( struct fwsocket \* sock ) [read]

Implementation of "listen" for DTLSv1.

#### Warning

Do not call this directly.

#### Parameters

<i>sock</i>	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](#).

```
{
    struct ssldata *ssl = sock->ssl;
    struct ssldata *newssl;
    struct fwsocket *newsock;
    union sockstruct client;
#ifdef __WIN32__
    int on = 1;
#else
    /* unsigned long on = 1; */
#endif

    if (!(newssl = objalloc(sizeof(*newssl), free_ssldata))) {
        return NULL;
    }

    newssl->flags |= SSL_DTLSCON;

    dtlssetopts(newssl, ssl, sock);
    memset(&client, 0, sizeof(client));
    if (DTLSv1_listen(newssl->ssl, &client) <= 0) {
        objunref(newssl);
        return NULL;
    }

    objlock(sock);
    if (!(newsock = make_socket(sock->addr.sa.sa_family, sock
        ->type, sock->proto, newssl))) {
        objunlock(sock);
```

```

        objunref(newssl);
        return NULL;
    }
    objunlock(sock);
    memcpy(&newsock->addr, &client, sizeof(newsock->addr));
#ifdef __WIN32__
    setsockopt(newsock->sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
#endif
#ifdef SO_REUSEPORT
    setsockopt(newsock->sock, SOL_SOCKET, SO_REUSEPORT, &on, sizeof(on));
#endif
#else
    /* ioctlsocket(newsock->sock, FIONBIO, (unsigned long*)&on); */
#endif
    objlock(sock);
    bind(newsock->sock, &sock->addr.sa, sizeof(sock->addr));
    objunlock(sock);
    connect(newsock->sock, &newsock->addr.sa, sizeof(newsock->addr));

    dtlsaccept(newsock);
    setflag(newsock, SOCK_FLAG_SSL);

    return (newsock);
}

```

### 12.6.5.2 void dtlshandltimeout ( struct fwsocket \* sock )

Handle DTLSv1 timeout.

#### Parameters

<i>sock</i>	Reference to socket.
-------------	----------------------

Definition at line 846 of file `sslutil.c`.

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

```

{
    if (!sock->ssl) {
        return;
    }

    objlock(sock->ssl);
    DTLSv1_handle_timeout(sock->ssl->ssl);
    objunlock(sock->ssl);
}

```

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

Get DTLSv1 timeout setting to default timeout.

#### Warning

Do not call this directly.

#### Parameters

<i>sock</i>	Reference to socket.
<i>timeleft</i>	timeval to store timeleft or set to default.
<i>defusec</i>	Default timeout to set.

Definition at line 831 of file `sslutil.c`.

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

```

{

```

```

    if (!sock || !sock->ssl || !sock->ssl->ssl) {
        return;
    }

    objlock(sock->ssl);
    if (!DTLSv1_get_timeout(sock->ssl->ssl, timeleft)) {
        timeleft->tv_sec = 0;
        timeleft->tv_usec = defusec;
    }
    objunlock(sock->ssl);
}

```

#### 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

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

Definition at line 325 of file [sslutil.c](#).

References [ssldata::ctx](#), [ssldata::ssl](#), and [SSL\\_DTLSV1](#).

Referenced by [sockettest\(\)](#).

```

{
    const SSL_METHOD *meth = DTLSv1_method();
    struct ssldata *ssl;

    ssl = sslinit(cacert, cert, key, verify, meth, SSL_DTLSV1);
    /* XXX BIO_CTRL_DGRAM_MTU_DISCOVER*/
    SSL_CTX_set_read_ahead(ssl->ctx, 1);

    return (ssl);
}

```

#### 12.6.5.5 void dtls\_serveropts ( struct fwsocket \* sock )

Start up the DTLSv1 Server.

##### Warning

This should not be called directly

##### See Also

[socketserver](#)

##### Parameters

<i>sock</i>	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\(\)](#).

{

```

    struct ssldata *ssl = sock->ssl;

    if (!ssl) {
        return;
    }

    dtlssetopts(ssl, NULL, sock);

    objlock(ssl);
    SSL_CTX_set_cookie_generate_cb(ssl->ctx, generate_cookie);
    SSL_CTX_set_cookie_verify_cb(ssl->ctx, verify_cookie);
    SSL_CTX_set_session_cache_mode(ssl->ctx, SSL_SESS_CACHE_OFF);

    SSL_set_options(ssl->ssl, SSL_OP_COOKIE_EXCHANGE);
    ssl->flags |= SSL_SERVER;
    objunlock(ssl);
}

```

#### 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

<i>data</i>	Referenece to the SSL data of socket.
<i>sock</i>	Socket FD to wait for data on.

Definition at line 179 of file [sslutil.c](#).

References [objlock\(\)](#), [objunlock\(\)](#), and [ssldata::ssl](#).

```

{
    struct ssldata *ssl = data;
    int ret, selfd, cnt = 0;

    if (!ssl) {
        return;
    }

    objlock(ssl);

    while (ssl->ssl && (ret = _ssl_shutdown(ssl) && (cnt < 3))) {
        selfd = socket_select(sock, ret);
        if (selfd <= 0) {
            break;
        }
        cnt++;
    }

    if (ssl->ssl) {
        SSL_free(ssl->ssl);
        ssl->ssl = NULL;
    }
    objunlock(ssl);
}

```

#### 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\(\)](#).

```

    {
        SSL_library_init();
        SSL_load_error_strings();
        OpenSSL_add_ssl_algorithms();

        if ((cookie_secret = malloc(COOKIE_SECRET_LENGTH)) {
            genrand(cookie_secret, COOKIE_SECRET_LENGTH)
        }
    }
}

```

#### 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

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

Definition at line 299 of file [sslutil.c](#).

References [SSL\\_SSLV2](#).

```

    {
        const SSL_METHOD *meth = SSLv2_method();

        return (sslinit(cacert, cert, key, verify, meth, SSL_SSLV2));
    }

```

#### 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

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

Definition at line 311 of file [sslutil.c](#).

References [ssldata::ssl](#), and [SSL\\_SSLV3](#).

Referenced by [sockettest\(\)](#).

```

    {
        const SSL_METHOD *meth = SSLv3_method();
        struct ssldata *ssl;

        ssl = sslinit(cacert, cert, key, verify, meth, SSL_SSLV3);

        return (ssl);
    }

```

#### 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

<i>sock</i>	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\(\)](#).

```

    if (!sock || !sock->ssl || (sock->ssl->flags & SSL_SERVER
    )) {
        return;
    }

    switch(sock->type) {
        case SOCK_DGRAM:
            dtlsconnect(sock);
            break;
        case SOCK_STREAM:
            sslsockstart(sock, NULL, 0);
            break;
    }
}

```

#### 12.6.5.11 void tlaccept ( struct fwsocket \* sock, struct ssldata \* orig )

Create SSL session for new connection.

##### Warning

This should never be called.

##### Parameters

<i>sock</i>	Reference too new incoming socket.
<i>orig</i>	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\(\)](#).

```

    {
        setflag(sock, SOCK_FLAG_SSL);
        if ((sock->ssl = objalloc(sizeof(*sock->ssl), free_ssldata))
        {
            sslsockstart(sock, orig, 1);
        }
    }
}

```



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

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

Definition at line 287 of file [sslutil.c](#).

References [SSL\\_TLSV1](#).

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

## 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 socket 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* ) [read]

Create a client thread on the socket.

It is not recommended 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

<i>sock</i>	Path to UNIX socket
<i>protocol</i>	Either SOCK_STREAM or SOCK_DGRAM, SOCK_STREAM is recommended.
<i>read</i>	Call back to call when read is ready.
<i>data</i>	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\(\)](#).

```

{
    struct fwsocket *fws;
    union sockstruct caddr, *saddr;
    char *temp = NULL;
    const char *tmpsock;
    int salen;
    mode_t omask;

    /*Create a UNIX socket structure*/
    if (!(fws = make_socket(PF_UNIX, protocol, 0, NULL))) {
        return NULL;
    }

    /* bind my endpoint to temp file*/
    if (protocol == SOCK_DGRAM) {
        /*yip i want only a inode here folks*/
        omask = umask(S_IXUSR | S_IRUSR | S_IWUSR | S_IWGRP | S_IRGRP | S_IXGRP
| S_IWOTH | S_IROTH | S_IXOTH);
        tmpsock = basename((char*)sock);
        temp = tempnam(NULL, tmpsock);
        if (strlenzero(temp)) {
            if (temp) {
                free(temp);
            }
            objunref(fws);
            return NULL;
        }

        /*Allocate address and connect to the client*/
        salen = sizeof(caddr.un);
        memset(&caddr.un, 0, salen);
        caddr.un.sun_family = PF_UNIX;
        strncpy((char *)caddr.un.sun_path, temp, sizeof(caddr.un.sun_path) -1);

        if (bind(fws->sock, (struct sockaddr *)&caddr.un, salen)) {
            /*reset umask*/
            umask(omask);
            if (temp) {
                if (!strlenzero(temp)) {
                    unlink(temp);
                }
                free(temp);
            }
            objunref(fws);
            return NULL;
        }
        /*reset umask*/
        umask(omask);
    }

    /*Allocate address and connect to the server*/
    saddr = &fws->saddr;
    salen = sizeof(saddr->un);
    memset(&saddr->un, 0, salen);
    saddr->un.sun_family = PF_UNIX;
    strncpy((char *)saddr->un.sun_path, sock, sizeof(saddr->un.sun_path) -1
);

    if (connect(fws->sock, (struct sockaddr *)&saddr->un, salen)) {
        if (temp) {
            if (!strlenzero(temp)) {
                unlink(temp);
            }
            free(temp);
        }
        objunref(fws);
        return NULL;
    }

    fws->flags |= SOCK_FLAG_UNIX;
    if (!(new_unixclientthread(fws, temp, read, data))) {
        if (temp) {
            if (!strlenzero(temp)) {
                unlink(temp);
            }
            free(temp);
        }
        objunref(fws);
        return NULL;
    }

    return (objref(fws)) ? fws : NULL;
}

```

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

Create and run UNIX server socket thread.

#### Parameters

<i>sock</i>	Path to UNIX socket.
<i>protocol</i>	Protocol number.
<i>mask</i>	Umask for the socket.
<i>read</i>	Callback to call when there is data available.
<i>data</i>	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\(\)](#).

```

{
    struct unixserv_sockthread *unsock;

    if (!(unsock = objalloc(sizeof(*unsock), free_unixserv))) {
        return NULL;
    }

    strncpy(unsock->sockpath, sock, UNIX_PATH_MAX);
    unsock->mask = mask;
    unsock->read = read;
    unsock->protocol = protocol;
    unsock->data = (objref(data)) ? data : NULL;

    /*Create a UNIX socket structure*/
    if (!(unsock->sock = make_socket(PF_UNIX, protocol,
        0, NULL))) {
        objunref(unsock);
        return NULL;
    }

    framework_mkthread(unsock_serv, NULL, NULL, unsock, 0);
    return (objref(unsock->sock)) ? unsock->sock : NULL;
}

```

## 12.8 Multicast sockets

Support for multicast sockets either IPv4 or IPv6.

### Functions

- void [mcast6\\_ip](#) (struct in6\_addr \*addr)  
*Randomly assign a SSM Multicast address.  
param addr Ip address structure to fill out.*
- void [mcast4\\_ip](#) (struct in\_addr \*addr)  
*Randomly 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 )

Randomly assign a SSM Multicast address.

#### Parameters

<i>addr</i>	Ip address structure to fill out.
-------------	-----------------------------------

Definition at line 504 of file [iputil.c](#).

References [genrand\(\)](#).

Referenced by [mcast\\_socket\(\)](#).

```

uint32_t mip, rand;
{
    do {
        rand = genrand(&mip, 3);
        mip >>= 8;
    } while (!rand || !(mip >> 8));
    mip |= 232 << 24;
    addr->s_addr = htonl(mip);
}

```

#### 12.8.2.2 void mcast6\_ip ( struct in6\_addr \* addr )

Randomly 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\(\)](#).

```

int mip, rand;
{

```

```

uint32_t *i;

#ifdef __WIN32
    i = (uint32_t*)&addr->s6_addr32;
#else
    i = (uint32_t*)&addr->u.Word;
#endif
    i[0] = htonl(0xFF350000);
    i[1] = 0;
    i[2] = 0;
    i[3] = 1 << 31;

    do {
        rand = genrand(&mip, 4);
    } while (!rand);

    i[3] = htonl(i[3] | mip);
}

```

**12.8.2.3 struct fwsocket\* mcast\_socket ( const char \* iface, int family, const char \* mcastip, const char \* port, int flags )**  
[read]

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

<i>iface</i>	Interface to send and recieve multicast traffic.
<i>family</i>	IP address family PF_INET or PF_INET6.
<i>mcastip</i>	Multicast ip to use must be in "family".
<i>port</i>	Port to use.
<i>flags</i>	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](#).

```

{
    struct fwsocket *fws;
    struct addrinfo hint, *result, *rp;
    struct in_addr *srcif;
    const char *srcip;
    int ifidx;
    int on = 1;
    int off = 0;
    int ttl = 50;
    socklen_t slen = sizeof(union sockstruct);
#ifdef __WIN32
    struct ifinfo *ifinfo;
#endif

    memset(&hint, 0, sizeof(hint));
    hint.ai_family = PF_UNSPEC;
    hint.ai_socktype = SOCK_DGRAM;
    hint.ai_protocol = IPPROTO_UDP;

#ifdef __WIN32
    if (!(srcip = get_ifipaddr(iface, family))) {

```

```

        return NULL;
    }

    if (getaddrinfo(srcip, port, &hint, &result) || !result) {
        free((void*)srcip);
        return NULL;
    }
    free((void*)srcip);
#else
    if (!(ifinf = get_ifinfo(iface))) {
        return NULL;
    }
    ifidx = ifinf->idx;

    srcip = (family == AF_INET) ? ifinf->ipv4addr : ifinf->ipv6addr
    ;
    if (!srcip || (getaddrinfo(srcip, port, &hint, &result) || !result)) {
        objunref(ifinf);
        return NULL;
    }
    objunref(ifinf);
#endif

    for(rp = result; rp; rp = result->ai_next) {
        if (!(fws = make_socket(rp->ai_family, rp->ai_socktype, rp->
            ai_protocol, NULL))) {
            continue;
        }
        break;
    }

    if (!rp || !fws) {
        freeaddrinfo(result);
        return NULL;
    }

    if(setsockopt(fws->sock, SOL_SOCKET, SO_REUSEADDR, (char*)&on, sizeof(
        on))) {
        objunref(fws);
        freeaddrinfo(result);
        return NULL;
    }

    if (rp->ai_family == PF_INET) {
        struct in_addr mcastip4;
        struct ip_mreq mg;
        struct sockaddr_in *src_ip;

        src_ip = (struct sockaddr_in*)rp->ai_addr;

        if (setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_TTL, (char*)&ttl
            , sizeof(ttl))) {
            objunref(fws);
            freeaddrinfo(result);
            return NULL;
        }

        if (flags && setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_LOOP, (
            char*)&off, sizeof(off))) {
            freeaddrinfo(result);
            objunref(fws);
            return NULL;
        }

        if (mcastip) {
            inet_lookup(PF_INET, mcastip, &mcastip4, sizeof(mcastip4)
        ));
        } else {
            seedrand();
            mcast4_ip(&mcastip4);
        }

        mg.imr_multiaddr = mcastip4;
        mg.imr_interface.s_addr = src_ip->sin_addr.s_addr;
        if (setsockopt(fws->sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char*)&mg
            , sizeof(mg))) {
            objunref(fws);
            freeaddrinfo(rp);
            return NULL;
        }

        memset(&srcif, 0, sizeof(srcif));
        srcif = &src_ip->sin_addr;
        if(setsockopt(fws->sock, IPPROTO_IP, IP_MULTICAST_IF, (char*)srcif,
            sizeof(*srcif))) {
            freeaddrinfo(rp);
            objunref(fws);

```

```

        return NULL;
    }
    src_ip->sin_addr.s_addr = mcastip4.s_addr;
} else if (rp->ai_family == PF_INET6) {
    struct in6_addr mcastip6;
    struct ipv6_mreq mg;
    struct sockaddr_in6 *src_ip;

#ifdef __WIN32
    ifidx = get_iface_index(iface);
#endif

    src_ip = (struct sockaddr_in6*)rp->ai_addr;

    if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_HOPS, (char
*)&tttl, sizeof(tttl)) {
        objunref(fws);
        freeaddrinfo(result);
        return NULL;
    }

    if (flags && setsockopt(fws->sock, IPPROTO_IPV6,
IPV6_MULTICAST_LOOP, (char*)&off, sizeof(off)) {
        freeaddrinfo(result);
        objunref(fws);
        return NULL;
    }

    if (mcastip) {
        inet_lookup(PF_INET6, mcastip, &mcastip6, sizeof(
mcastip6));
    } else {
        seedrand();
        mcast6_ip(&mcastip6);
    }

    mg.ipv6mr_multiaddr = mcastip6;
    mg.ipv6mr_interface = ifidx;
    if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_JOIN_GROUP, (char*)&mg
, sizeof(mg)) {
        objunref(fws);
        freeaddrinfo(rp);
        return NULL;
    }

    if (setsockopt(fws->sock, IPPROTO_IPV6, IPV6_MULTICAST_IF, (char*)&
ifidx, sizeof(ifidx)) {
        objref(fws);
        freeaddrinfo(rp);
        return NULL;
    }

    src_ip->sin6_addr = mcastip6;
}

if (bind(fws->sock, (struct sockaddr*)rp->ai_addr, sizeof(struct
sockaddr_storage)) {
    freeaddrinfo(result);
    objunref(fws);
    return NULL;
}

getsockname(fws->sock, &fws->addr.sa, &slen);
freeaddrinfo(result);
fws->flags |= SOCK_FLAG_MCAST;

return fws;
}

```



## 12.9 Linux network interface functions

Implement various interface routines from libnetlink.

### Files

- file [interface.c](#)  
*Wrapper around 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\\_ROUTEABLE](#) = 1 << 2 }  
*Order of precedence of ipv4.*
- enum [ipv6\\_score](#) { [IPV6\\_SCORE\\_RESERVED](#) = 1 << 0, [IPV6\\_SCORE\\_SIXIN4](#) = 1 << 1, [IPV6\\_SCORE\\_ROUTEABLE](#) = 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 precedence 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`.

```
{
    IPV4_SCORE_ZEROCONF = 1 << 0,
    IPV4_SCORE_RESERVED = 1 << 1,
    IPV4_SCORE_ROUTABLE = 1 << 2
};
```

#### 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`.

```
{
    IPV6_SCORE_RESERVED = 1 << 0,
    IPV6_SCORE_SIXIN4 = 1 << 1,
    IPV6_SCORE_ROUTABLE = 1 << 2
};
```

### 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\(\)](#).

```

    {
        if (nlh) {
            objunref(nlh);
        }
    }

```

#### 12.9.3.2 int create\_kernmac ( char \* ifname, char \* macdev, unsigned char \* mac )

Create a kernal MAC VLAN.

##### Parameters

<i>ifname</i>	Interface name to create
<i>macdev</i>	Base interface
<i>mac</i>	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\(\)](#).

```

    {
        struct iplink_req *req;
        struct rtattr *data, *linkinfo;
        unsigned char lmac[ETH_ALEN];
        char *type = "macvlan";
        int ifindex, ret;

        if (strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
            strlenzero(macdev) || (strlen(macdev) > IFNAMSIZ) ||
            (!objref(nlh) && !(nlh = nlhandle(0)))) {
            return (-1);
        }

        /*set the index of base interface*/
        if (!ifindex = get_iface_index(ifname)) {
            objunref(nlh);
            return (-1);
        }

        if (!mac) {
            randhwaddr(lmac);
        } else {
            strncpy((char *)lmac, (char *)mac, ETH_ALEN);
        }

        if (!(req = objalloc(sizeof(*req), NULL))) {
            objunref(nlh);
            return (-1);
        }

        req->n.nmsg_len = NMSG_LENGTH(sizeof(struct ifinfomsg));
        req->n.nmsg_type = RTM_NEWLINK;
        req->n.nmsg_flags = NLM_F_CREATE | NLM_F_EXCL | NLM_F_REQUEST;

        /*config base/dev/mac*/
        addattr_l(&req->n, sizeof(*req), IFLA_LINK, &ifindex, 4);
        addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, macdev, strlen(macdev));
    }

```

```

addattr_l(&req->n, sizeof(*req), IFLA_ADDRESS, lmac, ETH_ALEN);

/*type*/
linkinfo = NLMSG_TAIL(&req->n);
addattr_l(&req->n, sizeof(*req), IFLA_LINKINFO, NULL, 0);
addattr_l(&req->n, sizeof(*req), IFLA_INFO_KIND, type, strlen(type));

/*mode*/
data = NLMSG_TAIL(&req->n);
addattr_l(&req->n, sizeof(*req), IFLA_INFO_DATA, NULL, 0);
addattr32(&req->n, sizeof(*req), IFLA_MACVLAN_MODE, MACVLAN_MODE_PRIVATE);
data->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)data;
linkinfo->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)linkinfo;

objlock(nlh);
ret = rtnl_talk(nlh, &req->n, 0, 0, NULL);
objunlock(nlh);

objunref(nlh);
objunref(req);

return (ret);
}

```

### 12.9.3.3 int create\_kernvlan ( char \* ifname, unsigned short vid )

Create a VLAN on a interface.

#### Parameters

<i>ifname</i>	Interface to add VLAN to.
<i>vid</i>	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\(\)](#).

```

{
    struct iplink_req *req;
    char iface[IFNAMSIZ+1];
    struct rtattr *data, *linkinfo;
    char *type = "vlan";
    int ifindex, ret;

    if (strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ) ||
        (!objref(nlh) && !(nlh = nlhandle(0)))) {
        return (-1);
    }

    /*set the index of base interface*/
    if (!(ifindex = get_iface_index(ifname))) {
        objunref(nlh);
        return (-1);
    }

    if (!(req = objalloc(sizeof(*req), NULL))) {
        objunref(nlh);
        return (-1);
    }

    snprintf(iface, IFNAMSIZ, "%s.%i", ifname, vid);
    req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
    req->n.nlmsg_type = RTM_NEWLINK;
    req->n.nlmsg_flags = NLM_F_CREATE | NLM_F_EXCL | NLM_F_REQUEST;

    /*config base/dev/mac*/
    addattr_l(&req->n, sizeof(*req), IFLA_LINK, &ifindex, sizeof(ifindex));
    addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, iface, strlen(iface));

    /*type*/
    linkinfo = NLMSG_TAIL(&req->n);
    addattr_l(&req->n, sizeof(*req), IFLA_LINKINFO, NULL, 0);
    addattr_l(&req->n, sizeof(*req), IFLA_INFO_KIND, type, strlen(type));
}

```

```

/*vid*/
data = NLMSG_TAIL(&req->n);
addattr_l(&req->n, sizeof(*req), IFLA_INFO_DATA, NULL, 0);
addattr_l(&req->n, sizeof(*req), IFLA_VLAN_ID, &vid, sizeof(vid));

data->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)data;
linkinfo->rta_len = (char *)NLMSG_TAIL(&req->n) - (char *)linkinfo;

objlock(nlh);
ret = rtnl_talk(nlh, &req->n, 0, 0, NULL);
objunlock(nlh);

objunref(nlh);
objunref(req);

return (ret);
}

```

#### 12.9.3.4 int create\_tun ( const char \* ifname, const unsigned char \* hwaddr, int flags )

Create a tunnel device.

##### Parameters

<i>ifname</i>	Interface name to create..
<i>hwaddr</i>	Hardware address to assign (optionally).
<i>flags</i>	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\(\)](#).

```

{
    struct ifreq ifr;
    int fd, ifindex;
    char *tundev = "/dev/net/tun";

    /* open the tun/tap clone dev*/
    if ((fd = open(tundev, O_RDWR)) < 0) {
        return (-1);
    }

    /* configure the device*/
    memset(&ifr, 0, sizeof(ifr));
    ifr.ifr_flags = flags;
    strncpy(ifr.ifr_name, ifname, IFNAMSIZ);
    if (ioctl(fd, TUNSETIFF, (void *)&ifr) < 0 ) {
        perror("ioctl(TUNSETIFF) failed\n");
        close(fd);
        return (-1);
    }

    if (!(ifindex = get_iface_index(ifname))) {
        return (-1);
    }

    /* set the MAC address*/
    if (hwaddr) {
        set_interface_addr(ifindex, hwaddr);
    }

    /*set the network dev up*/
    set_interface_flags(ifindex, IFF_UP | IFF_RUNNING |
        IFF_MULTICAST | IFF_BROADCAST, 0);

    return (fd);
}

```

### 12.9.3.5 int delete\_kernmac ( char \* ifname )

Delete Kernel MAC VLAN.

#### Parameters

<i>ifname</i>	Interface to delete.
---------------	----------------------

#### Returns

-1 on error.

Definition at line 272 of file [interface.c](#).

```
    {  
        return (delete_interface(ifname));  
    }
```

### 12.9.3.6 int delete\_kernvlan ( char \* ifname, int vid )

Delete a VLAN.

#### Parameters

<i>ifname</i>	Interface we deleting vlan from.
<i>vid</i>	VLAN id to delete.

#### Returns

-1 on error.

Definition at line 201 of file [interface.c](#).

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

### 12.9.3.7 int get\_iface\_index ( const char \* ifname )

Get the netlink interface for a named interface.

#### Parameters

<i>ifname</i>	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\(\)](#).

```

{
    int ifindex;

    if (!objref(nlh) && !(nlh = nlhandle(0))) {
        return (0);
    }

    objlock(nlh);
    ll_init_map(nlh, 1);
    objunlock(nlh);

    ifindex = ll_name_to_index(ifname);

    objunref(nlh);
    return (ifindex);
}

```

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

Find best IP address for a interface.

**Todo** WIN32 Support

#### Parameters

<i>iface</i>	Interface name.
<i>family</i>	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\(\)](#).

```

{
    struct ifaddrs *ifaddr, *ifa;
    struct sockaddr_in *ipv4addr;
    int score = 0, nscore, iflen;
    uint32_t subnet = 0, match;
    char host[NI_MAXHOST] = "", tmp[NI_MAXHOST];

    if (!iface || getifaddrs(&ifaddr) == -1) {
        return NULL;
    }

    for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
        iflen = strlen(iface);
        if ((ifa->ifa_addr == NULL) || strncmp(ifa->ifa_name, iface, iflen) ||
            (ifa->ifa_addr->sa_family != family)) {
            continue;
        }

        /* Match aliases not vlans */
        if ((strlen(ifa->ifa_name) > iflen) && (ifa->ifa_name[iflen] != ':')) {
            continue;
        }

        switch (ifa->ifa_addr->sa_family) {
            case AF_INET:
                /* Find best ip address for a interface lowest priority is
                given to zeroconf then reserved ip's
                * finally find hte ip with shortest subnet bits */
                ipv4addr = (struct sockaddr_in*)ifa->ifa_netmask;
                match = ntohl(~ipv4addr->sin_addr.s_addr);

                nscore = score_ipv4((struct sockaddr_in*)ifa->
                ifa_addr, tmp, NI_MAXHOST);

                /* match score and subnet */

```

```

        if ((nscore > score) || ((nscore == score) && (match > subnet)
    )) {
        score = nscore;
        subnet = match;
        strncpy(host, tmp, NI_MAXHOST);
    }
    break;
case AF_INET6:
    nscore = score_ipv6((struct sockaddr_in6*)ifa->
ifa_addr, tmp, NI_MAXHOST);

    if (nscore > score) {
        score = nscore;
        strncpy(host, tmp, NI_MAXHOST);
    }
    break;
}
}
freeifaddrs(ifaddr);
return (strlenzero(host)) ? NULL : strdup(host);
}

```

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

Set interface down.

#### Parameters

<i>ifname</i>	Interface name.
<i>flags</i>	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\(\)](#).

```

{
    int ifindex;

    /*down the device*/
    if (!(ifindex = get_iface_index(ifname))) {
        return (-1);
    }

    /*set the network dev up*/
    set_interface_flags(ifindex, 0, IFF_UP | IFF_RUNNING |
        flags);

    return (0);
}

```

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

Get MAC addr for interface.

#### Parameters

<i>ifname</i>	Interface name
<i>hwaddr</i>	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\(\)](#).

```

{
    int ifindex;

    if (!hwaddr || strlenzero(ifname) || (strlen(ifname) > IFNAMSIZ)
        || (!objref(nlh) && !(nlh = nlhandle(0)))) {
        return (-1);
    }

    /*set the index of base interface*/
    if (!(ifindex = get_iface_index(ifname))) {
        objunref(nlh);
        return (-1);
    }

    ll_index_to_addr(ifindex, hwaddr, ETH_ALEN);
    objunref(nlh);
    return (0);
}

```

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

Rename interface helper.

**Parameters**

<i>oldname</i>	Original name.
<i>newname</i>	New name.

**Returns**

0 on success.

Definition at line 571 of file [interface.c](#).

References [get\\_iface\\_index\(\)](#), [ifdown\(\)](#), and [set\\_interface\\_name\(\)](#).

```

{
    int ifindex;

    ifdown(oldname, 0);

    if (!(ifindex = get_iface_index(oldname))) {
        return (-1);
    }
    set_interface_name(ifindex, newname);

    return (0);
}

```

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

Set interface up.

**Parameters**

<i>ifname</i>	Interface name.
<i>flags</i>	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\(\)](#).

```

{
    int ifindex;

    /*down the device*/
    if (!(ifindex = get_iface_index(ifname))) {
        return (-1);
    }

    /*set the network dev up*/
    set_interface_flags(ifindex, IFF_UP | IFF_RUNNING |
        flags, 0);

    return (0);
}

```

**12.9.3.13 int interface\_bind ( char \* iface, int protocol )**

Bind to device fd may be a existing socket.

**Parameters**

<i>iface</i>	Interface to bind too.
<i>protocol</i>	Protocol to use.

**Returns**

-1 on error.

Definition at line 453 of file [interface.c](#).

References [get\\_iface\\_index\(\)](#), and [set\\_interface\\_flags\(\)](#).

```

{
    struct sockaddr_ll sll;
    int proto = htons(protocol);
    int fd, ifindex;

    /*set the network dev up*/
    if (!(ifindex = get_iface_index(iface))) {
        return (-1);
    }
    set_interface_flags(ifindex, IFF_UP | IFF_RUNNING, 0);

    /* open network raw socket */
    if ((fd = socket(PF_PACKET, SOCK_RAW, proto)) < 0) {
        return (-1);
    }

    /*bind to the interface*/
    memset(&sll, 0, sizeof(sll));
    sll.sll_family = PF_PACKET;
    sll.sll_protocol = proto;
    sll.sll_ifindex = ifindex;
    if (bind(fd, (struct sockaddr *)&sll, sizeof(sll)) < 0) {
        perror("bind failed");
        close(fd);
        return (-1);
    }

    return (fd);
}

```

**12.9.3.14 void randhwaddr ( unsigned char \* addr )**

create random MAC address

**Parameters**

<i>addr</i>	Buffer char[ETH_ALEN] filled with the new address.
-------------	--

Definition at line 485 of file [interface.c](#).

References [genrand\(\)](#).

Referenced by [create\\_kernmac\(\)](#).

```

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

```

**12.9.3.15 int set\_interface\_addr ( int ifindex, const unsigned char \* hwaddr )**

Set interface MAC addr.

**Parameters**

<i>ifindex</i>	Interface index.
<i>hwaddr</i>	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\(\)](#).

```

{
    struct iplink_req *req;

    if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
        return (-1);
    }

    if (!(req = objalloc(sizeof(*req), NULL))) {
        objunref(nlh);
        return (-1);
    }

    req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
    req->n.nlmsg_type = RTM_NEWLINK;
    req->n.nlmsg_flags = NLM_F_REQUEST;
    req->i.ifi_index = ifindex;

    /*config base/dev/mac*/
    addattr_l(&req->n, sizeof(*req), IFLA_ADDRESS, hwaddr, ETH_ALEN);

    objlock(nlh);
    rtnl_talk(nlh, &req->n, 0, 0, NULL);
    objunlock(nlh);

    objunref(nlh);
    objunref(req);
    return (0);
}

```



References [get\\_iface\\_index\(\)](#), [ipaddr\\_req::i](#), [ipaddr\\_req::n](#), [objalloc\(\)](#), [objlock\(\)](#), [objref\(\)](#), [objunlock\(\)](#), and [objunref\(\)](#).

```

{
    struct ipaddr_req *req;
    inet_prefix lcl;
    int ifindex, bcast;

    if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
        return (-1);
    }

    if (!(req = objalloc(sizeof(*req), NULL))) {
        objunref(nlh);
        return (-1);
    }

    /*set the index of base interface*/
    if (!(ifindex = get_iface_index(ifname))) {
        objunref(nlh);
        return (-1);
    }

    req->n.nlmmsg_len = NLMMSG_LENGTH(sizeof(struct ifaddrmsg));
    req->n.nlmmsg_type = RTM_NEWADDR;
    req->n.nlmmsg_flags = NLM_F_REQUEST | NLM_F_EXCL | NLM_F_CREATE;

    req->i.ifa_scope = RT_SCOPE_HOST;
    req->i.ifa_index = ifindex;

    get_prefix(&lcl, ipaddr, AF_UNSPEC);
    req->i.ifa_family = lcl.family;
    req->i.ifa_prefixlen = lcl.bitlen;

    addattr_l(&req->n, sizeof(*req), IFA_LOCAL, &lcl.data, lcl.bytelen);
    addattr_l(&req->n, sizeof(*req), IFA_ADDRESS, &lcl.data, lcl.bytelen);
    if (lcl.family == AF_INET) {
        bcast = htonl((1 << (32 - lcl.bitlen)) - 1);
        addattr32(&req->n, sizeof(*req), IFA_BROADCAST, lcl.data[0] | bcast);
    }

    objlock(nlh);
    rtnl_talk(nlh, &req->n, 0, 0, NULL);
    objunlock(nlh);

    objunref(nlh);
    objunref(req);
    return (0);
}

```

### 12.9.3.18 int set\_interface\_name ( int ifindex, const char \* name )

Rename interface.

#### Parameters

<i>ifindex</i>	Interface index.
<i>name</i>	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\(\)](#).

```

{
    struct iplink_req *req;

    if ((!objref(nlh) && !(nlh = nlhandle(0)))) {
        return (-1);
    }
}

```

```
if (!(req = objalloc(sizeof(*req), NULL))) {
    objunref(nlh);
    return (-1);
}

req->n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
req->n.nlmsg_type = RTM_NEWLINK;
req->n.nlmsg_flags = NLM_F_REQUEST;
req->i.ifi_index = ifindex;

addattr_l(&req->n, sizeof(*req), IFLA_IFNAME, name, strlen((char *)name));

objlock(nlh);
rtnl_talk(nlh, &req->n, 0, 0, NULL);
objunlock(nlh);

objunref(nlh);
objunref(req);
return (0);
}
```

## 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 \*)  
*Callback used when processing config files.*
- typedef void(\* [config\\_catcb](#) )(struct [bucket\\_list](#) \*, const char \*)  
*Callback 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 categories 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 Wrapper 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 *)`

Callback used when processing a category.

##### Parameters

<i>entries</i>	Bucket list containing entries.
<i>name</i>	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

<i>item</i>	Name of the entry.
<i>value</i>	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 *)`

Callback used when processing config files.

##### Parameters

<i>categories</i>	Bucket list of categories.
<i>filename</i>	The filename.
<i>filepath</i>	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

<i>categories</i>	Bucketlist from a config file containing categories.
<i>cat_cb</i>	Callback to call on each category.



Definition at line 383 of file [config.c](#).

References [bucketlist\\_callback\(\)](#).

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

### 12.10.3.2 void config\_entry\_callback ( struct bucket\_list \* *entries*, config\_entrycb *entry\_cb* )

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

See Also

[config\\_entrycb](#)

#### Parameters

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

Definition at line 365 of file [config.c](#).

References [bucketlist\\_callback\(\)](#).

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

### 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

<i>file_cb</i>	Callback to call.
----------------	-------------------

Definition at line 400 of file [config.c](#).

References [bucketlist\\_callback\(\)](#).

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

### 12.10.3.4 struct bucket\_loop\* get\_category\_loop ( const char \* *configname* ) [read]

Return a bucket loop to allow iterating over categories.

## Parameters

<i>configname</i>	Name assigned to the config file when calling <a href="#">process_config()</a> .
-------------------	--

## Returns

Bucket loop iterator.

Definition at line 341 of file [config.c](#).

References [get\\_config\\_file\(\)](#), [init\\_bucket\\_loop\(\)](#), and [objunref\(\)](#).

```

{
    struct bucket_loop *cloop;
    struct bucket_list *file;

    file = get_config_file(configname);
    cloop = init_bucket_loop(file);
    objunref(file);
    return (cloop);
}

```

### 12.10.3.5 struct bucket\_list\* get\_category\_next ( struct bucket\_loop \* cloop, char \* name, int len ) [read]

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

<i>cloop</i>	Iterator created with <a href="#">get_category_loop</a> .
<i>name</i>	Buffer where the category name is copied.
<i>len</i>	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\(\)](#).

```

{
    struct config_category *category;

    if (cloop && (category = next_bucket_loop(cloop))) {
        if (category->entries) {
            if (!objref(category->entries)) {
                objunref(category);
                return (NULL);
            }
            if (!strlenzero(name)) {
                strncpy(name, category->name, len);
            }
            objunref(category);
            return (category->entries);
        } else {
            objunref(category);
        }
    }
    return (NULL);
}

```

### 12.10.3.6 struct bucket\_list\* get\_config\_category ( const char \* configname, const char \* category ) [read]

Return a single category.

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

#### Parameters

<i>configname</i>	Name assigned to the config file when calling <a href="#">process_config()</a> .
<i>category</i>	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\(\)](#).

```
    {
    struct bucket_list *file;
    struct config_category *cat;

    file = get_config_file(configname);
    if (category) {
        cat = bucket_list_find_key(file, category);
    } else {
        cat = bucket_list_find_key(file, "default");
    }

    objunref(file);
    if (cat) {
        if (!objref(cat->entries)) {
            objunref(cat);
            return (NULL);
        }
        objunref(cat);
        return (cat->entries);
    } else {
        return (NULL);
    }
}
```

#### 12.10.3.7 struct config\_entry\* get\_config\_entry ( struct bucket\_list \* categories, const char \* item ) [read]

Find the entry in a config file.

#### Parameters

<i>categories</i>	Categories bucketlist.
<i>item</i>	Item to search for.

#### Returns

Reference to a entry.

Definition at line 408 of file [config.c](#).

References [bucket\\_list\\_find\\_key\(\)](#).

```
    {
    struct config_entry *entry;

    entry = bucket_list_find_key(categories, item);

    return (entry);
}
```

### 12.10.3.8 struct bucket\_list\* get\_config\_file ( const char \* *configname* ) [read]

Returns the categories bucket for a config file.

#### Parameters

<i>configname</i>	Name assigned to the config file when calling <a href="#">process_config()</a> .
-------------------	--

#### 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\(\)](#).

```

{
    struct config_file *file;

    if ((file = bucket_list_find_key(configfiles,
        configname))) {
        if (file->cat) {
            if (!objref(file->cat)) {
                objunref(file);
                return (NULL);
            }
            objunref(file);
            return (file->cat);
        }
        objunref(file);
    }
    return (NULL);
}

```

### 12.10.3.9 int process\_config ( const char \* *configname*, const char \* *configfile* )

Process a configfile into buckets.

#### Parameters

<i>configname</i>	Name of the configuration.
<i>configfile</i>	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\(\)](#).

```

{
    struct config_file *file;
    struct config_category *category = NULL;
    FILE *config;
    char line[256];
    char item[128];
    char value[128];
    char *tmp = (char *)&line;
    char *token;

    if (!configfiles) {
        initconfigfiles();
    }

    file = create_conf_file(configname, configfile);
    addtobucket(configfiles, file);
}

```

```

if (!(config = fopen(file->filepath, "r"))) {
    return (-1);
}

while(fgets(line, sizeof(line) - 1, config)) {
    if (!(tmp = filterconf(line, 3)) {
        continue;
    }

    /*this is a new category*/
    if ((token = strchr(tmp, '[') && (token == tmp)) {
        tmp++;
        token = strchr(tmp, ']');
        token[0] = '\0';
        tmp = trim(tmp);
        if (!strlenzero(tmp)) {
            if (category) {
                objunref(category);
            }
            category = create_conf_category(tmp);
            addtobucket(file->cat, category);
        }
        continue;
    }

    if (sscanf(tmp, "%[^=] %*[%] %[^\\n]", (char *)&item, (char *)&value) !=
2) {
        continue;
    }

    if (!category) {
        category = create_conf_category("default");
        addtobucket(file->cat, category);
    }

    add_conf_entry(category, trim(item), trim(value));
}
fclose(config);
if (category) {
    objunref(category);
}
if (file) {
    objunref(file);
}
return (0);
}

```

#### 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\(\)](#).

```

{
    if (configfiles) {
        objunref(configfiles);
    }
}

```

## 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  
*Authentication 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 declaration 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 Project
```

### 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**

Authentication 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

<i>packet</i>	Reference to radius packet.
<i>data</i>	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](#).

```

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

```

### 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

<i>ipaddr</i>	IP address or hostname of server.
<i>auth</i>	Athentification port.
<i>acct</i>	Accounting port.
<i>secret</i>	Shared secret.
<i>timeout</i>	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](#).

```

{
    struct radius_server *server;

    if ((server = objalloc(sizeof(*server), del_radserver))) {
        ALLOC_CONST(server->name, ipaddr);
        ALLOC_CONST(server->authport, auth);
        ALLOC_CONST(server->acctport, acct);
        ALLOC_CONST(server->secret, secret);
        if (!servers) {
            servers = create_bucketlist(0, hash_server);
        }
        server->id = bucket_list_cnt(servers);
        server->timeout = timeout;
        gettimeofday(&server->service, NULL);
        addtobucket(servers, server);
    }

    objunref(server);
}

```

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

Add a integer attribute too the packet.

#### Parameters

<i>packet</i>	Radius packet to add too.
<i>type</i>	Attribute been added.
<i>val</i>	Value to add.

Definition at line 149 of file [radius.c](#).

```

{
    unsigned int tval;

    tval = htonl(val);
    addradattr(packet, type, (unsigned char *)&tval, sizeof(tval));
}

```

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

Add a integer attribute too the packet.

## Parameters

<i>packet</i>	Radius packet to add too.
<i>type</i>	Attribute been added.
<i>ipaddr</i>	IP to add.

Definition at line 160 of file [radius.c](#).

```

{
    unsigned int tval;

    tval = inet_addr(ipaddr);
    addradattr(packet, type, (unsigned char *)&tval, sizeof(tval));
}

```

## 12.11.5.4 void addradattrstr ( struct radius\_packet \* packet, char type, char \* str )

Add a integer attribute too the packet.

## Parameters

<i>packet</i>	Radius packet to add too.
<i>type</i>	Attribute been added.
<i>str</i>	Value to add.

Definition at line 171 of file [radius.c](#).

```

addradattr(packet, type, (unsigned char *)str, strlen(str));
}

```

## 12.11.5.5 struct radius\_packet\* new\_radpacket ( unsigned char code ) [read]

Create a new radius packet.

## See Also

[RADIUS\\_CODE](#)

## Parameters

<i>code</i>	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](#).

```

struct radius_packet *packet;

if ((packet = malloc(sizeof(*packet))) {
    memset(packet, 0, sizeof(*packet));
    packet->len = RAD_AUTH_HDR_LEN;
    packet->code = code;
    genrand(&packet->token, RAD_AUTH_TOKEN_LEN

```

```
    }  
    }  
    return (packet);  
}
```

#### 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

<i>packet</i>	Radius packet.
---------------	----------------

##### Returns

Pointer to next attribute

Definition at line 627 of file [radius.c](#).

References [radius\\_packet::attrs](#).

```
    {  
        return (packet->attrs);  
    }
```

#### 12.11.5.7 unsigned char\* radius\_attr\_next ( struct radius\_packet \* packet, unsigned char \* attr )

Return next packet attribute.

##### Parameters

<i>packet</i>	Radius packet.
<i>attr</i>	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](#).

```
    {  
        int offset = (packet->len - RAD_AUTH_HDR_LEN) - (attr -  
            packet->attrs);  
        if (!(offset - attr[1])) {  
            return NULL;  
        }  
        return (attr + attr[1]);  
    }
```

#### 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

<i>packet</i>	Radius packet to send.
<i>userpass</i>	Userpassword if required (added last requires special processing)
<i>read_cb</i>	Callback to call when response arrives.
<i>cb_data</i>	Reference to pass to callback.

## Returns

0 on success.

Definition at line 452 of file [radius.c](#).

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

## 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

<i>message</i>	String to encode.
<i>nonl</i>	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\(\)](#).

```

    {
        return b64enc_buf(message, strlen(message), nonl);
    }

```

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

Base 64 encode a buffer.

#### Parameters

<i>message</i>	Buffer to encode.
<i>len</i>	Length of the buffer.
<i>nonl</i>	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\(\)](#).

```

    {
        BIO *bmem, *b64;
        BUF_MEM *ptr;
        char *buffer;
        double encodedSize;

        encodedSize = 1.36*len;
        buffer = objalloc(encodedSize+1, NULL);

        b64 = BIO_new(BIO_f_base64());
        bmem = BIO_new(BIO_s_mem());
        b64 = BIO_push(b64, bmem);
        if (nonl) {
            BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL);
        }
        BIO_write(b64, message, len);
        BIO_flush(b64);
        BIO_get_mem_ptr(b64, &ptr);

        buffer = objalloc(ptr->length+1, NULL);
        memcpy(buffer, ptr->data, ptr->length);

        BIO_free_all(b64);
    }

```

```

    return buffer;
}

```

#### 12.12.2.3 uint16\_t checksum ( const void \* data, int len )

Obtain the checksum for a buffer.

##### Parameters

<i>data</i>	Buffer to create checksum of.
<i>len</i>	Buffer length.

##### Returns

Chechsum of data.

Definition at line 452 of file [util.c](#).

Referenced by [ipv4checksum\(\)](#), [ipv4icmpchecksum\(\)](#), [ipv4tcpchecksum\(\)](#), [ipv4udpchecksum\(\)](#), and [rfc6296\\_map\\_add\(\)](#).

```

{
    return (_checksum(data, len, 0));
}

```

#### 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

<i>checksum</i>	Checksum to add to generated checksum.
<i>data</i>	Buffer to create checksum of.
<i>len</i>	Buffer length.

##### Returns

Chechsum of data.

Definition at line 463 of file [util.c](#).

Referenced by [ipv4tcpchecksum\(\)](#), and [ipv4udpchecksum\(\)](#).

```

{
    return (_checksum(data, len, ~checksum));
}

```

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

Generate random sequence.

##### Parameters

<i>buf</i>	Buffer to write random data.
<i>len</i>	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\(\)](#).

```

    {
        return (RAND_bytes(buf, len));
    }

```

**12.12.2.6 char\* ltrim ( char \* *str* )**

Trim white space at the beginning of a string.

**Parameters**

<i>str</i>	String to trim.
------------	-----------------

**Returns**

Pointer to trimmed string.

Definition at line 353 of file [util.c](#).

References [strlenzero\(\)](#).

Referenced by [trim\(\)](#).

```

    {
        char *cur = str;

        if (strlenzero(str)) {
            return (str);
        }

        while (isspace(cur[0])) {
            cur++;
        }

        return (cur);
    }

```

**12.12.2.7 char\* rtrim ( const char \* *str* )**

Trim white space at the end of a string.

**Parameters**

<i>str</i>	String to trim.
------------	-----------------

**Returns**

Pointer to trimmed string.

Definition at line 372 of file [util.c](#).

References [strlenzero\(\)](#).

Referenced by [trim\(\)](#).

```

    {

```

```

int len;
char *cur = (char *)str;

if (strlenzero(str)) {
    return (cur);
}

len = strlen(str) - 1;
while(len && isspace(cur[len])) {
    cur[len] = '\0';
    len--;
}

return (cur);
}

```

#### 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\(\)](#).

```

{
int fd = open("/dev/random", O_RDONLY);
int len;
char buf[64];

len = read(fd, buf, 64);
RAND_seed(buf, len);
}

```

#### 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

<i>str</i>	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\(\)](#).

```

{
if (str && strlen(str)) {
    return (0);
}
return (1);
}

```

#### 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

<i>filename</i>	File to create.
<i>user</i>	User ID to set ownership.
<i>group</i>	Group ID to set ownership.

Definition at line 484 of file [util.c](#).

References [touch\(\)](#).

Referenced by [touch\(\)](#), and [xslt\\_apply\(\)](#).

```

    int res;
#else
extern void touch(const char *filename) {
#endif
    int fd;

    fd = creat(filename, 0600);
    close(fd);
#ifdef __WIN32__
    res = chown(filename, user, group);
    res++;
#endif
    return;
}

```

#### 12.12.2.11 char\* trim ( const char \* str )

Trim whitesapce from the beggining and end of a string.

#### Parameters

<i>str</i>	String to trim.
------------	-----------------

#### Returns

Trimed string.

Definition at line 393 of file [util.c](#).

References [ltrim\(\)](#), and [rtrim\(\)](#).

Referenced by [process\\_config\(\)](#).

```

    char *cur = (char *)str;
    cur = ltrim(cur);
    cur = rtrim(cur);
    return (cur);
}

```

#### 12.12.2.12 uint64\_t tvtontp64 ( struct timeval \* tv )

Convert a timeval struct to 64bit NTP time.

#### Parameters

<i>tv</i>	Timeval struct to convert.
-----------	----------------------------

**Returns**

64 bit NTP time value.

Definition at line 405 of file [util.c](#).

Referenced by [get\\_ip6\\_addrprefix\(\)](#).

```
    {  
    return (((uint64_t)tv->tv_sec + 2208988800u) << 32) + ((uint32_t)tv->  
        tv_usec * 4294.967296));  
    }
```

**12.12.2.13 uint16\_t verifysum ( const void \* *data*, int *len*, const uint16\_t *check* )**

Verify a checksum.

**Parameters**

<i>data</i>	Data to generate checksum.
<i>len</i>	Length of data.
<i>check</i>	Checksum to check against.

**Returns**

0 when checksum is verified.

Definition at line 473 of file [util.c](#).

```
    {  
    return (_checksum(data, len, check));  
    }
```

## 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.*
- [SHA2-256 Hashing 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

<i>digest1</i>	Digest to compare.
<i>digest2</i>	Digest to compare.

##### Returns

0 on equality.

Definition at line [223](#) of file [util.c](#).

```

    {
        return (_digest_cmp(digest1, digest2, 16));
    }

```

#### 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

<i>buff</i>	HMAC returned in this buffer (16 bytes).
<i>data</i>	Data to sign.
<i>len</i>	Length of data.
<i>key</i>	Key to signwith.
<i>klen</i>	Length of key.

Definition at line 290 of file [util.c](#).

References [md5sum2\(\)](#).

```

    {
        _hmac(buff, data, len, key, klen, md5sum2, 16);
    }

```

#### 12.14.2.3 void md5sum ( unsigned char \* buff, const void \* data, unsigned long len )

Calculate the MD5 hash.

##### Parameters

<i>buff</i>	buffer to place the hash (16 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.

Definition at line 202 of file [util.c](#).

References [md5sum2\(\)](#).

```

    {
        md5sum2(buff, data, len, NULL, 0);
    }

```

#### 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

<i>buff</i>	buffer to place the hash (16 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.
<i>data2</i>	Second data chunk to calculate.
<i>len2</i>	Length of data2.

Definition at line 185 of file [util.c](#).

Referenced by [md5hmac\(\)](#), and [md5sum\(\)](#).

```

    {
        MD5_CTX c;

        MD5_Init(&c);
        MD5_Update(&c, data, len);
        if (data2) {
            MD5_Update(&c, data2, len2);
        }
        MD5_Final(buff, &c);
    }

```

## 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

<i>digest1</i>	Digest to compare.
<i>digest2</i>	Digest to compare.

##### Returns

0 on equality.

Definition at line [233](#) of file [util.c](#).

```

    {
        return (_digest_cmp(digest1, digest2, 20));
    }

```

#### 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

<i>buff</i>	HMAC returned in this buffer (20 bytes).
<i>data</i>	Data to sign.
<i>len</i>	Length of data.
<i>key</i>	Key to signwith.
<i>klen</i>	Length of key.



Definition at line 302 of file [util.c](#).

References [sha1sum2\(\)](#).

```

    {
        _hmac(buff, data, len, key, klen, sha1sum2, 20);
    }

```

#### 12.15.2.3 void sha1sum ( unsigned char \* buff, const void \* data, unsigned long len )

Calculate the SHA1 hash.

##### Parameters

<i>buff</i>	buffer to place the hash (20 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.

Definition at line 173 of file [util.c](#).

References [sha1sum2\(\)](#).

```

    {
        sha1sum2(buff, data, len, NULL, 0);
    }

```

#### 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

<i>buff</i>	buffer to place the hash (20 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.
<i>data2</i>	Second data chunk to calculate.
<i>len2</i>	Length of data2.

Definition at line 156 of file [util.c](#).

Referenced by [get\\_ip6\\_addrprefix\(\)](#), [sha1hmac\(\)](#), and [sha1sum\(\)](#).

```

    {
        SHA_CTX c;

        SHA_Init(&c);
        SHA_Update(&c, data, len);
        if (data2) {
            SHA_Update(&c, data2, len2);
        }
        SHA_Final(buff, &c);
    }

```

## 12.16 SHA2-256 Hashing 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

<i>digest1</i>	Digest to compare.
<i>digest2</i>	Digest to compare.

##### Returns

0 on equality.

Definition at line [243](#) of file [util.c](#).

```

}
return (_digest_cmp(digest1, digest2, 32));
{
```

#### 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

<i>buff</i>	HMAC returned in this buffer (32 bytes).
<i>data</i>	Data to sign.
<i>len</i>	Length of data.
<i>key</i>	Key to signwith.
<i>klen</i>	Length of key.

Definition at line 314 of file [util.c](#).

References [sha256sum2\(\)](#).

```

    {
        _hmac(buff, data, len, key, klen, sha256sum2, 32);
    }

```

#### 12.16.2.3 void sha256sum ( unsigned char \* buff, const void \* data, unsigned long len )

Calculate the SHA2-256 hash.

##### Parameters

<i>buff</i>	buffer to place the hash (32 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.

Definition at line 144 of file [util.c](#).

References [sha256sum2\(\)](#).

```

    {
        sha256sum2(buff, data, len, NULL, 0);
    }

```

#### 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

<i>buff</i>	buffer to place the hash (32 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.
<i>data2</i>	Second data chunk to calculate.
<i>len2</i>	Length of data2.

Definition at line 127 of file [util.c](#).

Referenced by [sha256hmac\(\)](#), and [sha256sum\(\)](#).

```

    {
        SHA256_CTX c;

        SHA256_Init(&c);
        SHA256_Update(&c, data, len);
        if (data2) {
            SHA256_Update(&c, data2, len2);
        }
        SHA256_Final(buff, &c);
    }

```

## 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

<i>digest1</i>	Digest to compare.
<i>digest2</i>	Digest to compare.

##### Returns

0 on equality.

Definition at line [253](#) of file [util.c](#).

```

}
    return (_digest_cmp(digest1, digest2, 64));
{

```

#### 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

<i>buff</i>	HMAC returned in this buffer (64 bytes).
<i>data</i>	Data to sign.
<i>len</i>	Length of data.
<i>key</i>	Key to signwith.
<i>klen</i>	Length of key.

Definition at line 326 of file [util.c](#).

References [sha512sum2\(\)](#).

```

    {
        _hmac(buff, data, len, key, klen, sha512sum2, 64);
    }

```

### 12.17.2.3 void sha512sum ( unsigned char \* buff, const void \* data, unsigned long len )

Calculate the SHA2-512 hash.

#### Parameters

<i>buff</i>	buffer to place the hash (64 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.

Definition at line 114 of file [util.c](#).

References [sha512sum2\(\)](#).

```

    {
        sha512sum2(buff, data, len, NULL, 0);
    }

```

### 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

<i>buff</i>	buffer to place the hash (64 bytes).
<i>data</i>	First data chunk to calculate.
<i>len</i>	Length of data.
<i>data2</i>	Second data chunk to calculate.
<i>len2</i>	Length of data2.

Definition at line 97 of file [util.c](#).

Referenced by [sha512hmac\(\)](#), and [sha512sum\(\)](#).

```

    {
        SHA512_CTX c;
        SHA512_Init(&c);
        SHA512_Update(&c, data, len);
        if (data2) {
            SHA512_Update(&c, data2, len2);
        }
        SHA512_Final(buff, &c);
    }

```

## 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](#).

```
{  
  IP\_PROTO\_V4 = 4,  
  IP\_PROTO\_V6 = 6  
};
```

### 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 return the IP address.

##### Parameters

<i>family</i>	Protocol family either PF_INET or PF_INET6.
<i>host</i>	Hostname or IP address to lookup.
<i>addr</i>	A structure in_addr or in6_addr the result is returned in.
<i>len</i>	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()`.

```

{
    struct addrinfo hint, *result, *ainfo;
    int ret = 0;

    memset(&hint, 0, sizeof(hint));
    hint.ai_family = family;

    if (getaddrinfo(host, NULL, &hint, &result) || !result) {
        return ret;
    }

    for(ainfo = result; ainfo; ainfo = ainfo->ai_next) {
        switch(ainfo->ai_family) {
            case PF_INET:
                if (len >= sizeof(struct in_addr)) {
                    struct sockaddr_in *sa4 = (struct sockaddr_in*)ainfo->
ai_addr;
                    memcpy(addr, &sa4->sin_addr, len);
                    ret = 1;
                }
                break;
            case PF_INET6:
                if (len >= sizeof(struct in6_addr)) {
                    struct sockaddr_in6 *sa6 = (struct sockaddr_in6*)ainfo->
ai_addr;
                    memcpy(addr, &sa6->sin6_addr, len);
                    ret = 1;
                }
                break;
        }
        if (ret) {
            break;
        }
    }
    freeaddrinfo(result);
    return ret;
}

```

#### 12.18.3.2 `int packetchecksum ( uint8_t * pkt )`

Generic IPv4 and IPv6 Checksum.

##### Parameters

<i>pkt</i>	Packet buffer to check.
------------	-------------------------

## Returns

Checksum.

Definition at line 208 of file [iputil.c](#).

References [IP\\_PROTO\\_V4](#), [IP\\_PROTO\\_V6](#), and [packetchecksumv4\(\)](#).

```
    {  
    struct iphdr *ip = (struct iphdr *)pkt;  
    switch(ip->version) {  
        case IP_PROTO_V4:  
            return (packetchecksumv4(pkt));  
            break;  
        case IP_PROTO_V6:  
            break;  
    }  
    return (-1);  
}
```



## 12.19 IPv4 functions

Helper functions for various calculations.

### Files

- file [iputil.c](#)  
*IPv4 And IPv6 Utilities.*

### 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 address.*
- 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 checksum of a IPv4 ICMP packet.*
- void [ipv4checksum](#) (uint8\_t \*pkt)  
*Set the checksum 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

<i>ip</i>	Network address to check against.
<i>cidr</i>	Number of bits in the subnet.
<i>test</i>	IP address to check

#### Returns

0 if test is not in the network ip/cidr.

Definition at line 456 of file [iputil.c](#).

```

uint32_t ip1, ip2;
{
#ifdef __WIN32
    inet_pton(AF_INET, ip, &ip1);
    inet_pton(AF_INET, test, &ip2);
#else
    ip1 = inet_addr(ip);
    ip2 = inet_addr(test);
#endif

    ip1 = ntohl(ip1) >> (32-cidr);
    ip2 = ntohl(ip2) >> (32-cidr);

    if (!(ip1 ^ ip2)) {
        return 1;
    } else {
        return 0;
    }
}

```

### 12.19.2.2 `uint32_t cidrcnt ( int bitlen )`

Return the number of IP addresses in a given bitmask.

#### Parameters

<i>bitlen</i>	Subnet bits (CIDR).
---------------	---------------------

#### Returns

Number of IP addresses including network and broadcast address.

Definition at line 372 of file [iputil.c](#).

```

{
    if (bitlen) {
        return pow(2, (32-bitlen));
    } else {
        return 0xFFFFFFFF;
    }
}

```

**12.19.2.3** `const char* cidrtosn ( int bitlen, char * buf, int size )`

Return the dotted quad notation subnet mask from a CIDR.

**Parameters**

<i>bitlen</i>	Subnet length bits.
<i>buf</i>	Buffer to copy the subnet address too.
<i>size</i>	Size of buffer.

**Returns**

pointer to buffer on success or NULL.

Definition at line 228 of file [iputil.c](#).

```

{
    uint32_t nm;
    uint8_t *nmb = (uint8_t*)&nm;

    if (!buf) {
        return NULL;
    }

    if (bitlen) {
        nm = ~(1 << (32-bitlen))-1;
    } else {
        nm = 0;
    }

    snprintf(buf, size, "%i.%i.%i.%i", nmb[3], nmb[2], nmb[1], nmb[0]);
    return buf;
}

```

**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**

<i>ipaddr</i>	Network address.
<i>cidr</i>	CIDR subnet bit length.
<i>buf</i>	Buffer to copy address too.
<i>size</i>	Length of buffer.

**Returns**

Pointer to buffer or NULL on error.

Definition at line 319 of file [iputil.c](#).

```

{
    uint32_t ip, mask;
    uint8_t *ipb = (uint8_t*)&ip;

#ifdef __WIN32
    inet_pton(AF_INET, ipaddr, &ip);
#else
    ip = inet_addr(ipaddr);
#endif
}

```

```

    if (cidr) {
        mask = (1 << (32-cidr))-1;
        ip = ntohl(ip);
        ip = (ip & ~mask) | mask;
    } else {
        ip = 0;
    }
    snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
    return buf;
}

```

#### 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

<i>ipaddr</i>	Network address.
<i>cidr</i>	Bits in the subnet mask.
<i>buf</i>	Buffer that the result is placed in.
<i>size</i>	Length of buffer.

##### Returns

Pointer to *buf* with the result copied to *buf*.

Definition at line 286 of file [iputil.c](#).

```

    {
        uint32_t ip;
        uint8_t *ipb = (uint8_t*)&ip;

        if (!buf) {
            return NULL;
        }

#ifdef __WIN32
        inet_pton(AF_INET, ipaddr, &ip);
#else
        ip = inet_addr(ipaddr);
#endif
        if (cidr) {
            ip = ntohl(ip);
            ip = ip & ~((1 << (32-cidr))-1);
            ip++;
        } else {
            ip = 1;
        }

        snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
        return buf;
    }

```

#### 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

<i>ipaddr</i>	Network address.
<i>cidr</i>	Bits in the subnet mask.
<i>buf</i>	Buffer that the result is placed in.
<i>size</i>	Length of buffer.

## Returns

Pointer to buf with the result copied to buf.

Definition at line 347 of file [iputil.c](#).

```

    {
        uint32_t ip, mask;
        uint8_t *ipb = (uint8_t*)&ip;

#ifdef __WIN32
        inet_pton(AF_INET, ipaddr, &ip);
#else
        ip = inet_addr(ipaddr);
#endif
        if (cidr) {
            mask = (1 << (32-cidr))-1;
            ip = ntohl(ip);
            ip = (ip & ~mask) | mask;
            ip--;
        } else {
            ip = 0;
        }
        snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
        return buf;
    }

```

### 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

<i>ipaddr</i>	Ipaddr to calculate for
<i>cidr</i>	Length of the subnet bitmask.
<i>buf</i>	Buffer that the result is placed in.
<i>size</i>	Length of buffer.

## Returns

Pointer to buf with the result copied to buf.

Definition at line 254 of file [iputil.c](#).

```

    {
        uint32_t ip;
        uint8_t *ipb = (uint8_t*)&ip;

        if (!buf) {
            return NULL;
        }

#ifdef __WIN32
        inet_pton(AF_INET, ipaddr, &ip);

```

```

#else
    ip = inet_addr(ipaddr);
#endif
    if (cidr) {
        ip = ntohl(ip);
        ip = ip & ~(1 << (32-cidr))-1;
    } else {
        ip = 0;
    }

    snprintf(buf, size, "%i.%i.%i.%i", ipb[3], ipb[2], ipb[1], ipb[0]);
    return buf;
}

```

#### 12.19.2.8 void ipv4checksum ( uint8\_t \* pkt )

Set the checksum of a IPv4 Packet.

##### Parameters

<i>pkt</i>	Packet to update.
------------	-------------------

Definition at line 154 of file [iputil.c](#).

References [checksum\(\)](#).

Referenced by [packetchecksumv4\(\)](#).

```

{
    struct iphdr *ip = (struct iphdr *)pkt;
    ip->check = 0;
    ip->check = checksum(ip, (4 * ip->ihl));
}

```

#### 12.19.2.9 void ipv4icmpchecksum ( uint8\_t \* pkt )

Set the checksum of a IPv4 ICMP packet.

##### Parameters

<i>pkt</i>	ICMP Packet to update.
------------	------------------------

Definition at line 143 of file [iputil.c](#).

References [checksum\(\)](#).

Referenced by [packetchecksumv4\(\)](#).

```

{
    struct iphdr *ip = (struct iphdr *)pkt;
    struct icmphdr *icmp = (struct icmphdr *) (pkt + (4 * ip->ihl));
    icmp->checksum = 0;
    icmp->checksum = checksum(icmp, ntohs(ip->tot_len) - (ip->ihl * 4));
}

```

#### 12.19.2.10 void ipv4tcpchecksum ( uint8\_t \* pkt )

Update the TCP checksum of a IPv4 packet.

##### Parameters

<i>pkt</i>	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()`.

```

{
    struct iphdr *ip = (struct iphdr *)pkt;
    struct tcphdr *tcp = (struct tcphdr *) (pkt + (4 * ip->ihl));
    uint16_t plen, csum;
    struct pseudohdr phdr;

    /* get tcp packet len*/
    plen = ntohs(ip->tot_len) - (4 * ip->ihl);
    tcp->check = 0;
    phdr.saddr = ip->saddr;
    phdr.daddr = ip->daddr;
    phdr.zero = 0;
    phdr.proto = ip->protocol;
    phdr.len = htons(plen);
    csum = checksum(&phdr, sizeof(phdr));
    tcp->check = checksum_add(csum, tcp, plen);
}

```

#### 12.19.2.11 void ipv4udpchecksum ( uint8\_t \* pkt )

Update the UDP checksum of a IPv4 packet.

##### Parameters

<i>pkt</i>	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()`.

```

{
    struct iphdr *ip = (struct iphdr *)pkt;
    struct udphdr *udp = (struct udphdr *) (pkt + (4 * ip->ihl));
    uint16_t csum, plen;
    struct pseudohdr phdr;

    /* get tcp packet len*/
    plen = ntohs(ip->tot_len) - (4 * ip->ihl);
    udp->check = 0;
    phdr.saddr = ip->saddr;
    phdr.daddr = ip->daddr;
    phdr.zero = 0;
    phdr.proto = ip->protocol;
    phdr.len = htons(plen);
    csum = checksum(&phdr, sizeof(phdr));
    udp->check = checksum_add(csum, udp, plen);
}

```

#### 12.19.2.12 int packetchecksumv4 ( uint8\_t \* pkt )

Update the checksum of a IPv4 packet.

##### Parameters

<i>pkt</i>	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\(\)](#).

```

    struct iphdr *ip = (struct iphdr *)pkt;
    ipv4checksum(pkt);
    switch(ip->protocol) {
        case IPPROTO_ICMP:
            ipv4icmpchecksum(pkt);
            break;
        case IPPROTO_TCP:
            ipv4tcpchecksum(pkt);
            break;
        case IPPROTO_UDP:
            ipv4udpchecksum(pkt);
            break;
        default:
            return (-1);
    }
    return (0);
}

```

**12.19.2.13 int reservedip ( const char \* ipaddr )**

Check IP against list of reserved IP's.

**Parameters**

<i>ipaddr</i>	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\(\)](#).

```

    uint32_t ip;
#ifdef __WIN32
    inet_pton(PF_INET, ipaddr, &ip);
#else
    ip = inet_addr(ipaddr);
#endif
    ip = ntohl(ip);
    if (!((0xe0000000 ^ ip) >> 28)) { /* 224/4 */
        return 1;
    } else if (!((0x00000000 ^ ip) >> 24)) { /* 0/8 */
        return 1;
    } else if (!((0x0a000000 ^ ip) >> 24)) { /* 10/8 */
        return 1;
    } else if (!((0x7f000000 ^ ip) >> 24)) { /* 127/8 */
        return 1;
    } else if (!((0x64400000 ^ ip) >> 22)) { /* 100.64/10 */
        return 1;
    } else if (!((0xac100000 ^ ip) >> 20)) { /* 172.16/12 */
        return 1;
    } else if (!((0xc6120000 ^ ip) >> 17)) { /* 198.18/15 */
        return 1;
    } else if (!((0xc0a80000 ^ ip) >> 16)) { /* 192.168/16 */
        return 1;
    }
}

```



```

    } else if (!(0xa9fe0000 ^ ip) >> 16)) { /* 169.254/16 */
        return 1;
    } else if (!(0xc0000200 ^ ip) >> 8)) { /* 192.0.2/24 */
        return 1;
    } else if (!(0xc6336400 ^ ip) >> 8)) { /* 198.51.100/24 */
        return 1;
    } else if (!(0xcb007100 ^ ip) >> 8)) { /* 203.0.113/24 */
        return 1;
    }
    return 0;
}

```

#### 12.19.2.14 int score\_ipv4 ( struct sockaddr\_in \* sa4, char \* ipaddr, int iplen )

Return a score for a IPv4 address.

##### Note

This does not follow the RFC as getaddrinfo would.

##### Parameters

<i>sa4</i>	Socket addr to check.
<i>ipaddr</i>	Buffer to place IP address.
<i>iplen</i>	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\(\)](#).

```

                                                                    {
uint32_t addr;
int nscore;

addr = sa4->sin_addr.s_addr;

/* Get ipaddr string*/
inet_ntop(AF_INET, &sa4->sin_addr, ipaddr, iplen);

/* Score the IP*/
if (!(0xa9fe0000 ^ ntohl(addr)) >> 16)) {
    nscore = IPV4_SCORE_ZEROCONF;
} else if (reservedip(ipaddr)) {
    nscore = IPV4_SCORE_RESERVED;
} else {
    nscore = IPV4_SCORE_ROUTABLE;
}

return nscore;
}

```

## 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 checksum 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

<i>ipaddr</i>	To check.
<i>network</i>	Network to check against.
<i>bits</i>	Network length.

#### Returns

0 if the ipaddr is in the network.

Definition at line 47 of file [iputil.c](#).

```

{
    uint8_t cnt, bytelen, bitlen;
    uint32_t mask, res = 0;
    uint32_t *nw = (uint32_t *)network;
    uint32_t *ip = (uint32_t *)ipaddr;

    /*calculate significant bytes and bits outside boundry*/
    if ((bitlen = bits % 32)) {

```

```

        bytelen = (bits - bitlen) / 32;
        bytelen++;
    } else {
        bytelen = bits / 32;
    }

    /*end loop on first mismatch do not check last block*/
    for(cnt = 0; (!res && (cnt < (bytelen - 1))); cnt++) {
        res += nw[cnt] ^ ip[cnt];
    }

    /*process last block if no error sofar*/
    if (!res) {
        mask = (bitlen) ? htonl(~((1 << (32 - bitlen)) - 1)) : -1;
        res += (nw[cnt] & mask) ^ (ip[cnt] & mask);
    }

    return (res);
}

```

### 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

<i>mac48</i>	Buffer containing MAC address 6 bytes.
<i>eui64</i>	Buffer that will be written with address 8bytes.

Definition at line 668 of file [interface.c](#).

Referenced by [get\\_ip6\\_addrprefix\(\)](#).

```

        {
            eui64[0] = (mac48[0] & 0xFE) ^ 0x02; /*clear multicast bit and flip local
            assignment*/
            eui64[1] = mac48[1];
            eui64[2] = mac48[2];
            eui64[3] = 0xFF;
            eui64[4] = 0xFE;
            eui64[5] = mac48[3];
            eui64[6] = mac48[4];
            eui64[7] = mac48[5];
        }

```

### 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

<i>iface</i>	External system interface name.
<i>prefix</i>	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\(\)](#).

```

uint64_t ntpts;
unsigned char eui64[8];
unsigned char sha1[20];
unsigned char mac48[ETH_ALEN];
struct timeval tv;

if (ifhwaddr(iface, mac48)) {
    return (-1);
}

gettimeofday(&tv, NULL);
ntpts = tvtontp64(&tv);

eui48to64(mac48, eui64);
shalsum2(sha1, (void *)&ntpts, sizeof(ntpts), (void *)eui64, sizeof
(eui64));

prefix[0] = 0xFD; /*0xFC | 0x01 FC00/7 with local bit set [8th bit]*/
memcpy(prefix + 1, sha1+15, 5); /*LSD 40 bits of the SHA hash*/

return (0);
}

```

#### 12.20.2.4 char\* ipv6to4prefix ( const char \* ipaddr )

Return IPv6 to IPv4 Prefix for the address.

##### Parameters

<i>ipaddr</i>	IPv4 Address to obtain mapping for
---------------	------------------------------------

##### Returns

6to4 Address prefix.

Definition at line 427 of file [iputil.c](#).

```

uint32_t ip;
uint8_t *ipa;
char *pre6;

#ifdef __WIN32
if (!inet_pton(AF_INET, ipaddr, &ip)) {
    return NULL;
}
#else
if (!(ip = inet_addr(ipaddr))) {
    return NULL;
}
#endif

pre6 = malloc(10);
ipa=(uint8_t*)&ip;
snprintf(pre6, 10, "%02x%02x:%02x%02x", ipa[0], ipa[1], ipa[2], ipa[3]);
return pre6;
}

```

#### 12.20.2.5 int packetchecksumv6 ( uint8\_t \* pkt )

Prototype to check checksum on packet.

##### Parameters

<i>pkt</i>	Packet buffer to check.
------------	-------------------------

Definition at line 189 of file [iputil.c](#).

```

{

```

```

struct iphdr *ip = (struct iphdr *)pkt;
switch(ip->protocol) {
    case IPPROTO_ICMP:
        break;
    case IPPROTO_TCP:
        break;
    case IPPROTO_UDP:
        break;
    default:
        return (-1);
}
return (0);
}

```

### 12.20.2.6 int score\_ipv6 ( struct sockaddr\_in6 \* sa6, char \* ipaddr, int iplen )

Return a score for a IPv6 address.

#### Note

This does not follow the RFC as getaddrinfo would.

#### Parameters

<i>sa6</i>	Socket addr to check.
<i>ipaddr</i>	Buffer to place IP address.
<i>iplen</i>	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\(\)](#).

```

uint32_t *ipptr, match;
int nscore;

#ifdef __WIN32
    ipptr = sa6->sin6_addr.s6_addr32;
#else
    ipptr = (uint32_t*)sa6->sin6_addr.u.Word;
#endif
match = ntohl(ipptr[0]) >> 16;

/* exclude link local multicast and special addresses */
if (!(0xFE80 ^ match) || !(0xFF ^ (match >> 8)) || !match) {
    return 0;
}

/*Score ip private/sixin4/routable*/
if (!(0xFC ^ (match >> 9))) {
    nscore = IPV6_SCORE_RESERVED;
} else if (match == 2002) {
    nscore = IPV6_SCORE_SIXIN4;
} else {
    nscore = IPV6_SCORE_ROUTABLE;
}
inet_ntop(AF_INET6, ipptr, ipaddr, iplen);

return nscore;
}

```

## 12.21 File utility functions

Convincence wrappers around 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

Convincence wrappers around stat.

#### 12.21.2 Function Documentation

##### 12.21.2.1 int is\_dir ( const char \* *path* )

Determine if a path is a directory.

##### Parameters

<i>path</i>	Path of directory to check.
-------------	-----------------------------

##### Returns

1 if the path exists and is a directory 0 otherwise.

Definition at line 55 of file [fileutil.c](#).

```

{
    struct stat sr;
    if (!stat(path, &sr) && S_ISDIR(sr.st_mode)) {
        return 1;
    } else {
        return 0;
    }
}

```

##### 12.21.2.2 int is\_exec ( const char \* *path* )

Determine if a file is executable.

## Parameters

<i>path</i>	Path of file to check.
-------------	------------------------

## Returns

1 if the path exists and is executable 0 otherwise.

Definition at line 67 of file [fileutil.c](#).

```

{
    struct stat sr;
    if (!stat(path, &sr) && (S_IXUSR & sr.st_mode)) {
        return 1;
    } else {
        return 0;
    }
}

```

12.21.2.3 int is\_file ( const char \* *path* )

Determine if a file exists.

## Parameters

<i>path</i>	Filename.
-------------	-----------

## Returns

1 if the file exists 0 otherwise.

Definition at line 43 of file [fileutil.c](#).

```

{
    struct stat sr;
    if (!stat(path, &sr)) {
        return 1;
    } else {
        return 0;
    }
}

```

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 permissions.

## Parameters

<i>dir</i>	Directory to create.
<i>mode</i>	Initial mode to set.
<i>user</i>	Initial UID.
<i>group</i>	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](#).

```

                                                                    {
#endif
    struct stat sr;

#ifdef __WIN32
    if ((stat(dir, &sr) && (errno == ENOENT)) && !mkdir(dir)) {
#else
    if ((stat(dir, &sr) && (errno == ENOENT)) && !mkdir(dir, mode) && !chown(
        dir, user, group)) {
#endif
        return 0;
    }
    return -1;
}
```



## 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 distinguished name linked list.*
- struct [ldap\\_attrval](#)  
*LDAP attribute value.*
- struct [ldap\\_attr](#)  
*LDAP attribute.*
- struct [ldap\\_entry](#)  
*LDAP entry.*
- struct [ldap\\_results](#)  
*LDAP results.*
- struct [sasl\\_defaults](#)  
*SASL Parameters used in authentication.*
- 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 declaration of structure.*
- typedef struct [ldap\\_modify](#) [ldap\\_modify](#)  
*Forward declaration of structure.*
- typedef struct [ldap\\_add](#) [ldap\\_add](#)  
*Forward declaration of structure.*

### Enumerations

- enum [ldap\\_starttls](#) { [LDAP\\_STARTTLS\\_NONE](#), [LDAP\\_STARTTLS\\_ATTEMPT](#), [LDAP\\_STARTTLS\\_ENFORCE](#) }
  - 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 requiring a distinguished 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\\_rematr](#) (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 declaration of structure.

Definition at line 789 of file [dtsapp.h](#).

#### 12.22.2.2 typedef struct [ldap\\_conn](#) [ldap\\_conn](#)

Forward declaration of structure.

Definition at line 785 of file [dtsapp.h](#).

#### 12.22.2.3 typedef struct [ldap\\_modify](#) [ldap\\_modify](#)

Forward declaration 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](#).

```

{
    LDAP\_ATTRTYPE\_CHAR,
    LDAP\_ATTRTYPE\_B64,
    LDAP\_ATTRTYPE\_OCTET
};

```

#### 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](#).

```

    {
        LDAP_STARTTLS_NONE,
        LDAP_STARTTLS_ATTEMPT,
        LDAP_STARTTLS_ENFORCE
    };

```

## 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

<i>ladd</i>	Reference to new DN structure.
<i>attr</i>	Attribute to add.
...	NULL terminated list of values.

### Returns

0 on success.

Definition at line 1500 of file [openldap.c](#).

References [objunref\(\)](#).

```

    {
        va_list a_list;
        char *val;
        struct ldap_modreq *modr;

        if (! (modr = getaddreq(ladd, attr))) {
            return 1;
        }

        va_start(a_list, attr);
        while ((val = va_arg(a_list, void *))) {
            if (add_modifyval(modr, val)) {
                objunref(modr);
                return(1);
            }
        }

        objunref(modr);
        va_end(a_list);
        return 0;
    }

```

12.22.4.2 `struct ldap_add* ldap_addinit ( const char * dn ) [read]`

Create a reference to add a new DN.

### Parameters

<i>dn</i>	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\(\)](#).

```

{
    struct ldap_add *mod;

    if (! (mod = objalloc(sizeof(*mod), free_add))) {
        return NULL;
    }

    ALLOC_CONST(mod->dn, dn);
    if (!mod->dn) {
        objunref(mod);
        return NULL;
    }

    if (! (mod->bl = create_bucketlist(4, modify_hash))) {
        objunref(mod);
        return NULL;
    }

    return mod;
}

```

**12.22.4.3** `struct ldap_conn* ldap_connect ( const char * uri, enum ldap_starttls starttls, int timelimit, int limit, int debug, int * err )` [read]

Connect to a LDAP server.

#### Parameters

<i>uri</i>	Server to connect too.
<i>starttls</i>	Starttls flags to disallow,allow or enforce SSL.
<i>timelimit</i>	Query timelimit.
<i>limit</i>	Results limit.
<i>debug</i>	Set LDAP_OPT_DEBUG_LEVEL and LBER_OPT_DEBUG_LEVEL to this level.
<i>err</i>	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](#).

```

{
    struct ldap_conn *ld;
    int version = 3;
    int res, sslres;
    struct timeval timeout;

    if (! (ld = objalloc(sizeof(*ld), free_ldapconn))) {
        return NULL;
    }

    ld->uri = strdup(uri);
    ld->sctrlsp = NULL;
    ld->timelim = timelimit;
    ld->limit = limit;
    ld->sasl = NULL;

    if ((res = ldap_initialize(&ld->ldap, ld->uri)) != LDAP_SUCCESS) {
        objunref(ld);
        ld = NULL;
    } else {
        if (debug) {
            ldap_set_option(NULL, LDAP_OPT_DEBUG_LEVEL, &debug);
            lber_set_option(NULL, LBER_OPT_DEBUG_LEVEL, &debug);
        }
    }
}

```

```

    }
    if (timelimit) {
        timeout.tv_sec = timelimit;
        timeout.tv_usec = 0;
        ldap_set_option(ld->ldap, LDAP_OPT_NETWORK_TIMEOUT, (void *)&
            timeout);
    }
    ldap_set_option(ld->ldap, LDAP_OPT_PROTOCOL_VERSION, &version);
    ldap_set_option(ld->ldap, LDAP_OPT_REFERRALS, (void *)LDAP_OPT_ON);
    ldap_set_rebind_proc(ld->ldap, ldap_rebind_proc, ld);

    if ((starttls != LDAP_STARTTLS_NONE) & !
        ldap_tls_inplace(ld->ldap) && (sslres = ldap_start_tls_s(ld->ldap, ld->sctrlsp,
            NULL))) {
        if (starttls == LDAP_STARTTLS_ENFORCE) {
            objunref(ld);
            ld = NULL;
            res = sslres;
        }
    }
}
*err = res;
return ld;
}

```

#### 12.22.4.4 int ldap\_doaddd ( struct ldap\_conn \* ld, struct ldap\_add \* ladd )

Write new DN to server.

##### Parameters

<i>ld</i>	Reference to connection to the LDAP server.
<i>ladd</i>	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](#).

```

{
    struct bucket_loop *bloop;
    struct ldap_modreq *modr;
    LDAPMod **modarr, **tmp, *item;
    int tot=0, res;

    tot = bucket_list_cnt(ladd->bl);
    tmp = modarr = calloc(sizeof(void *), (tot+1));

    bloop = init_bucket_loop(ladd->bl);
    while(bloop && (modr = next_bucket_loop(bloop))) {
        if (!(item = ldap_reqtoarr(modr, -1))) {
            ldap_mods_free(modarr, 1);
            return LDAP_NO_MEMORY;
        }
        *tmp = item;
        tmp++;
        objunref(modr);
    }
    objunref(bloop);
    *tmp = NULL;

    objlock(ld);
    res = ldap_modify_ext_s(ld->ldap, ladd->dn, modarr, ld->sctrlsp,
        NULL);
    objunlock(ld);
    ldap_mods_free(modarr, 1);

    return res;
}

```

#### 12.22.4.5 int ldap\_domodify ( struct ldap\_conn \* ld, struct ldap\_modify \* lmod )

Apply the modification to the server.

##### Parameters

<i>ld</i>	Reference to LDAP connection.
<i>lmod</i>	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\\_repatrr\(\)](#).

```

{
    struct bucket_loop *bloop;
    struct ldap_modreq *modr;
    LDAPMod **modarr, **tmp, *item;
    int cnt, tot=0, res;

    if (!objref(ld)) {
        return LDAP_UNAVAILABLE;
    }

    for(cnt = 0; cnt < 3; cnt++) {
        tot += bucket_list_cnt(lmod->bl[cnt]);
    }
    tmp = modarr = calloc(sizeof(void *), (tot+1));

    for(cnt = 0; cnt < 3; cnt++) {
        bloop = init_bucket_loop(lmod->bl[cnt]);
        while(bloop && (modr = next_bucket_loop(bloop))) {
            if (!(item = ldap_reqtoarr(modr, cnt))) {
                ldap_mods_free(modarr, 1);
                objunref(ld);
                return LDAP_NO_MEMORY;
            }
            *tmp = item;
            tmp++;
            objunref(modr);
        }
        objunref(bloop);
    }
    *tmp = NULL;

    objlock(ld);
    res = ldap_modify_ext_s(ld->ldap, lmod->dn, modarr, ld->sctrlsp,
        , NULL);
    objunlock(ld);
    ldap_mods_free(modarr, 1);
    objunref(ld);
    return res;
}

```

#### 12.22.4.6 const char\* ldap\_errmsg ( int res )

Return LDAP error for a ldap error.

##### Parameters

<i>res</i>	LDAP error id.
------------	----------------

**Returns**

Error string.

Definition at line 576 of file [openldap.c](#).

```
    {  
        return ldap_err2string(res);  
    }
```

**12.22.4.7 struct ldap\_attr\* ldap\_getattr ( struct ldap\_entry \* entry, const char \* attr ) [read]**

Find and return attribute in a entry.

**Parameters**

<i>entry</i>	Entry to return attribute from.
<i>attr</i>	Attribute to return.

**Returns**

Attribute reference matching attr.

Definition at line 1108 of file [openldap.c](#).

References [ldap\\_entry::attrs](#), and [bucket\\_list\\_find\\_key\(\)](#).

```
    {  
        if (!entry || !entry->attrs) {  
            return NULL;  
        }  
        return (struct ldap_attr *)bucket_list_find_key  
            (entry->attrs, attr);  
    }
```

**12.22.4.8 struct ldap\_entry\* ldap\_getentry ( struct ldap\_results \* results, const char \* dn ) [read]**

Find and return the entry from the results for a specific dn.

**Parameters**

<i>results</i>	Results to search in.
<i>dn</i>	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](#).

```
    {  
        if (!results || !dn) {  
            return NULL;  
        }  
        return (struct ldap_entry *)bucket_list_find_key  
            (results->entries, dn);  
    }
```



#### 12.22.4.9 int ldap\_mod\_add ( struct ldap\_modify \* lmod, const char \* attr, ... )

Add values to a attribute.

##### Parameters

<i>lmod</i>	LDAP modification reference.
<i>attr</i>	Attribute to modify.
<i>...</i>	Values to add.

##### Returns

Zero on success.

Definition at line 1238 of file [openldap.c](#).

References [objunref\(\)](#).

Referenced by [ldap\\_mod\\_addattr\(\)](#).

```

{
    va_list a_list;
    char *val;
    struct ldap_modreq *modr;

    if (!(modr = getmodreq(lmod, attr, LDAP_MOD_ADD))) {
        return 1;
    }

    va_start(a_list, attr);
    while((val = va_arg(a_list, void *))) {
        if (add_modifyval(modr, val)) {
            objunref(modr);
            return(1);
        }
    }

    objunref(modr);
    va_end(a_list);
    return 0;
}

```

#### 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

<i>ldap</i>	Reference to the connection.
<i>dn</i>	DN to remove values from.
<i>attr</i>	Attribute to add value to.
<i>value</i>	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\(\)](#).

```

{
    int res = 0;
    struct ldap_modify *lmod;

    if (!(lmod = ldap_modifyinit(dn))) {

```

```

        return LDAP_NO_MEMORY;
    }

    if (ldap_mod_add(lmod, attr, value, NULL)) {
        objunref(lmod);
        return LDAP_NO_MEMORY;
    }

    res = ldap_domodify(ldap, lmod);
    objunref(lmod);
    return res;
}

```

#### 12.22.4.11 int ldap\_mod\_del ( struct ldap\_modify \* lmod, const char \* attr, ... )

Delete values from a attribute.

##### Parameters

<i>lmod</i>	LDAP modification reference.
<i>attr</i>	Attribute to modify.
<i>...</i>	Values to remove.

##### Returns

Zero on success.

Definition at line 1211 of file [openldap.c](#).

References [objunref\(\)](#).

Referenced by [ldap\\_mod\\_delattr\(\)](#).

```

{
    va_list a_list;
    char *val;
    struct ldap_modreq *modr;

    if (!(modr = getmodreq(lmod, attr, LDAP_MOD_DELETE))) {
        return 1;
    }

    va_start(a_list, attr);
    while((val = va_arg(a_list, void *))) {
        if (add_modifyval(modr, val)) {
            objunref(modr);
            return(1);
        }
    }

    objunref(modr);
    va_end(a_list);
    return 0;
}

```

#### 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

<i>ldap</i>	Reference to the connection.
<i>dn</i>	DN to remove values from.
<i>attr</i>	Attribute to remove values from.
<i>value</i>	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\\_remattr\(\)](#).

```

    {
        struct ldap_modify *lmod;
        int res;

        if (!(lmod = ldap_modifyinit(dn))) {
            return LDAP_NO_MEMORY;
        }
        if (ldap_mod_del(lmod, attr, value, NULL)) {
            objunref(lmod);
            return LDAP_NO_MEMORY;
        }

        res = ldap_domodify(ldap, lmod);
        objunref(lmod);
        return res;
    }

```

#### 12.22.4.13 int ldap\_mod\_remattr ( struct ldap\_conn \* *ldap*, const char \* *dn*, const char \* *attr* )

Delete a attribute from a DN.

**Parameters**

<i>ldap</i>	Reference to the connection.
<i>dn</i>	DN to remove attribute from.
<i>attr</i>	Attribute to remove.

**Returns**

Non zero ldap error on failure

Definition at line 1406 of file [openldap.c](#).

References [ldap\\_mod\\_delattr\(\)](#).

```

    {
        return ldap_mod_delattr(ldap, dn, attr, NULL);
    }

```

#### 12.22.4.14 int ldap\_mod\_rep ( struct ldap\_modify \* *lmod*, const char \* *attr*, ... )

Replace a attribute.

**Parameters**

<i>lmod</i>	LDAP modification referenece.
<i>attr</i>	Attribute to modify.
...	Values to replace.

**Returns**

Zero on success.

Definition at line 1265 of file [openldap.c](#).

References [objunref\(\)](#).

Referenced by [ldap\\_mod\\_repatrr\(\)](#).

```

{
    va_list a_list;
    char *val;
    struct ldap_modreq *modr;

    if (!(modr = getmodreq(lmod, attr, LDAP_MOD_REPLACE))) {
        return 1;
    }

    va_start(a_list, attr);
    while((val = va_arg(a_list, void *))) {
        if (add_modifyval(modr, val)) {
            objunref(modr);
            return(1);
        }
    }

    objunref(modr);
    va_end(a_list);
    return 0;
}

```

**12.22.4.15** `int ldap_mod_repatrr ( struct ldap_conn * ldap, const char * dn, const char * attr, const char * value )`

Replace the value of a attribute in a DN.

**Parameters**

<i>ldap</i>	Reference to the connection.
<i>dn</i>	DN to replace attribute in.
<i>attr</i>	Attribute to replace.
<i>value</i>	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\(\)](#).

```

{
    struct ldap_modify *lmod;
    int res;

    if (!(lmod = ldap_modifyinit(dn))) {
        return LDAP_NO_MEMORY;
    }

    if (ldap_mod_rep(lmod, attr, value, NULL)) {
        objunref(lmod);
        return LDAP_NO_MEMORY;
    }

    res = ldap_domodify(ldap, lmod);
    objunref(lmod);
    return res;
}

```

#### 12.22.4.16 struct ldap\_modify\* ldap\_modifyinit ( const char \* *dn* ) [read]

Create a modification reference for a DN.

##### Parameters

<i>dn</i>	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\\_repatrr\(\)](#).

```

{
    struct ldap_modify *mod;
    int cnt;

    if (!mod = objalloc(sizeof(*mod), free_modify)) {
        return NULL;
    }

    ALLOC_CONST(mod->dn, dn);
    if (!mod->dn) {
        objunref(mod);
        return NULL;
    }

    for(cnt=0; cnt < 3; cnt++) {
        if (!mod->bl[cnt] = create_bucketlist(4,
            modify_hash)) {
            objunref(mod);
            return NULL;
        }
    }

    return mod;
}

```

#### 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

<i>ld</i>	Reference to LDAP connection.
<i>mech</i>	SASL mechanism.
<i>realm</i>	SASL realm.
<i>authcid</i>	SASL auth id.
<i>passwd</i>	Password for authid.
<i>authzid</i>	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::ld](#), [sasl\\_defaults::mech](#), [objalloc\(\)](#), [objlock\(\)](#), [objunlock\(\)](#), [objunref\(\)](#), [sasl\\_defaults::passwd](#), [sasl\\_defaults::realm](#), [ldap\\_conn::sasl](#), and [ldap\\_conn::sctrlsp](#).

```

{
    struct sasl_defaults *sasl;
    int res, sasl_flags = LDAP_SASL_AUTOMATIC | LDAP_SASL_QUIET;

    if (!objref(ld)) {
        return LDAP_UNAVAILABLE;
    }

    if (!(sasl = objalloc(sizeof(*sasl), free_sasl))) {
        return LDAP_NO_MEMORY;
    }

    ALLOC_CONST(sasl->passwd, passwd);

    if (mech) {
        ALLOC_CONST(sasl->mech, mech);
    } else {
        ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_MECH, &sasl->mech);
    }

    if (realm) {
        ALLOC_CONST(sasl->realm, realm);
    } else {
        ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_REALM, &sasl->realm);
    }

    if (authcid) {
        ALLOC_CONST(sasl->authcid, authcid);
    } else {
        ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHCID, &sasl->authcid);
    }

    if (authzid) {
        ALLOC_CONST(sasl->authzid, authzid);
    } else {
        ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHZID, &sasl->authzid);
    }

    objlock(ld);
    if (ld->sasl) {
        objunref(ld->sasl);
    }
    ld->sasl = sasl;
    res = ldap_sasl_interactive_bind_s(ld->ldap, NULL, sasl->mech, ld->
        sclrlsp, NULL, sasl_flags, dts_sasl_interact, sasl);
    objunlock(ld);
    objunref(ld);
    return res;
}

```

**12.22.4.18** `struct ldap_results* ldap_search_base ( struct ldap_conn * ld, const char * base, const char * filter, int b64enc, int * res, ... )` [read]

Search LDAP connection base.

#### Parameters

<i>ld</i>	Reference to LDAP connection.
<i>base</i>	Search base dn.
<i>filter</i>	Search filter.
<i>b64enc</i>	Base 64 encode attributes.
<i>res</i>	Pointer containing LDAP error.
<i>...</i>	NULL terminated list of attributes to include.

#### Returns

Search results structure.

Definition at line 669 of file [openldap.c](#).

References [\\_dtsldapsearch\(\)](#).

```

{
    va_list a_list;
    char *attr, **tmp, **attrs = NULL;
    int cnt = 1;

    va_start(a_list, res);
    while (( attr=va_arg(a_list, void *)) {
        cnt++;
    }
    va_end(a_list);

    if (cnt > 1) {
        tmp = attrs = malloc(sizeof(void *)*cnt);

        va_start(a_list, res);
        while (( attr=va_arg(a_list, char *)) {
            *tmp = attr;
            tmp++;
        }
        va_end(a_list);
        *tmp=NULL;
    }

    return _dtsldapsearch(ld, base, LDAP_SCOPE_BASE, filter,
        attrs, b64enc, res);
}

```

**12.22.4.19** `struct ldap_results* ldap_search_one ( struct ldap_conn * ld, const char * base, const char * filter, int b64enc, int * res, ... )` [read]

Search LDAP connection one level.

#### Parameters

<i>ld</i>	Reference to LDAP connection.
<i>base</i>	Search base dn.
<i>filter</i>	Search filter.
<i>b64enc</i>	Base 64 encode attributes.
<i>res</i>	Pointer containing LDAP error.
...	NULL terminated list of attributes to include.

#### Returns

Search results structure.

Definition at line 635 of file `openldap.c`.

References `_dtsldapsearch()`.

```

{
    va_list a_list;
    char *attr, **tmp, **attrs = NULL;
    int cnt = 1;

    va_start(a_list, res);
    while (( attr=va_arg(a_list, void *)) {
        cnt++;
    }
    va_end(a_list);

    if (cnt > 1) {
        tmp = attrs = malloc(sizeof(void *)*cnt);

        va_start(a_list, res);
        while (( attr=va_arg(a_list, char *)) {
            *tmp = attr;
            tmp++;
        }
        va_end(a_list);
        *tmp=NULL;
    }
}

```

```

return _dtsldapsearch(ld, base, LDAP_SCOPE_ONELEVEL, filter,
    attrs, b64enc, res);
}

```

**12.22.4.20** `struct ldap_results* ldap_search_sub ( struct ldap_conn * ld, const char * base, const char * filter, int b64enc, int * res, ... )` [read]

Search LDAP connection subtree.

#### Parameters

<i>ld</i>	Reference to LDAP connection.
<i>base</i>	Search base dn.
<i>filter</i>	Search filter.
<i>b64enc</i>	Base 64 encode attributes.
<i>res</i>	Pointer containing LDAP error.
...	NULL terminated list of attributes to include.

#### Returns

Search results structure.

Definition at line 601 of file [openldap.c](#).

References [\\_dtsldapsearch\(\)](#).

Referenced by [ldap\\_simplerebind\(\)](#).

```

{
    va_list a_list;
    char *attr, **tmp, **attrs = NULL;
    int cnt = 1;

    va_start(a_list, res);
    while ((attr=va_arg(a_list, void *))) {
        cnt++;
    }
    va_end(a_list);

    if (cnt > 1) {
        tmp = attrs = malloc(sizeof(void *)*cnt);

        va_start(a_list, res);
        while ((attr=va_arg(a_list, char *))) {
            *tmp = attr;
            tmp++;
        }
        va_end(a_list);
        *tmp=NULL;
    }

    return _dtsldapsearch(ld, base, LDAP_SCOPE_SUBTREE, filter,
        attrs, b64enc, res);
}

```

**12.22.4.21** `int ldap_simplebind ( struct ldap_conn * ld, const char * dn, const char * passwd )`

Bind to the connection with simple bind requiring a distinguished name and password.

#### Parameters

<i>ld</i>	LDAP connection to bind to.
<i>dn</i>	Distinguished name to bind with.
<i>passwd</i>	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()`.

```

    {
        struct ldap_simple *simple;
        struct berval *cred;
        int res, len = 0;

        if (!objref(ld)) {
            return LDAP_UNAVAILABLE;
        }

        if (passwd) {
            len = strlen(passwd);
        }
        simple = objalloc(sizeof(*simple), free_simple);
        cred = calloc(sizeof(*cred), 1);
        cred->bv_val = malloc(len);
        memcpy(cred->bv_val, passwd, len);
        cred->bv_len=len;
        simple->cred = cred;
        simple->dn = strdup(dn);

        objlock(ld);
        if (ld->simple) {
            objunref(ld->simple);
        }
        ld->simple = simple;
        res = ldap_sasl_bind_s(ld->ldap, simple->dn, LDAP_SASL_SIMPLE, simple
            ->cred, ld->sctrlsp, NULL, NULL);
        objunlock(ld);
        objunref(ld);
        return res;
    }

```

**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 distinguished name and password search for a user dn, bind to the connection with the retrieved dn and user password.

## Parameters

<i>ldap</i>	LDAP connection to bind too.
<i>initialdn</i>	Initial dn to bind with.
<i>initialpw</i>	Password for the initial dn.
<i>base</i>	Search base to find user.
<i>filter</i>	LDAP filter to apply to find user.
<i>uidrdn</i>	Attribute containing user id.
<i>uid</i>	To search and bind as.
<i>passwd</i>	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\(\)](#).

```

    {
        int res, flen;
        struct ldap_results *results;
        const char *sfilt;

        if (!objref(ldap)) {
            return LDAP_UNAVAILABLE;
        }

        if ((res = ldap_simplebind(ldap, initialdn, initialpw)) {
            objunref(ldap);
            return res;
        }

        flen=strlen(uidrdn) + strlen(filter) + strlen(uid) + 7;
        sfilt = malloc(flen);
        snprintf((char *)sfilt, flen, "(&(%s=%s)%s)", uidrdn, uid, filter);

        if (!(results = ldap_search_sub(ldap, base, sfilt, 0, &res,
            uidrdn, NULL))) {
            free((void *)sfilt);
            objunref(ldap);
            return res;
        }
        free((void *)sfilt);

        if (results->count != 1) {
            objunref(results);
            objunref(ldap);
            return LDAP_INAPPROPRIATE_AUTH;
        }

        res = ldap_simplebind(ldap, results->first_entry
            ->dn, passwd);
        objunref(ldap);
        objunref(results);
        return res;
    }

```

**12.22.4.23 void ldap\_unref\_attr ( struct ldap\_entry \* entry, struct ldap\_attr \* attr )**

Remove a attribute from a entry.

**Parameters**

<i>entry</i>	The entry to remove attr from.
<i>attr</i>	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\(\)](#).

```

{
    if (!entry || !attr) {
        return;
    }

    if (objcnt(attr) > 1) {
        objunref(attr);
    } else {
        if (attr == entry->first_attr) {
            entry->first_attr = attr->next;
        }
        remove_bucket_item(entry->attrs, attr);
    }
}

```

12.22.4.24 void ldap\_unref\_entry ( struct ldap\_results \* *results*, struct ldap\_entry \* *entry* )

Remove a entry from a result.

#### Parameters

<i>results</i>	The result to remove entry from.
<i>entry</i>	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\(\)](#).

```
    {  
    if (!results || !entry) {  
        return;  
    }  
  
    if (objcnt(entry) > 1) {  
        objunref(entry);  
    } else {  
        if (entry == results->first_entry) {  
            results->first_entry = entry->next;  
        }  
        remove_bucket_item(results->entries, entry);  
    }  
}
```

## 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 declaration of structure.*
- typedef struct [xml\\_search](#) [xml\\_search](#)  
*Forward declaration of structure.*
- typedef struct [xml\\_doc](#) [xml\\_doc](#)  
*Forward declaration 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 reference.*
- 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 declaration of structure.

Definition at line 631 of file [dtsapp.h](#).

### 12.23.2.2 typedef struct xml\_node xml\_node

Forward declaration of structure.

Definition at line 625 of file [dtsapp.h](#).

### 12.23.2.3 typedef struct xml\_search xml\_search

Forward declaration 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* ) [read]

Append a node to a path.

#### Parameters

<i>xmldoc</i>	Reference to XML document.
<i>xpath</i>	Path to add the node too.
<i>name</i>	Node name.
<i>value</i>	Node value.
<i>attrkey</i>	Attribute to create on node.
<i>keyval</i>	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\(\)](#).

```

{
    struct xml_node *newnode;
    xmlNodePtr parent;
    xmlNodePtr child;
    xmlChar *encval;

    if (!objref(xmldoc)) {
        return NULL;
    }

    objlock(xmldoc);
    if (!(parent = xml_getparent(xmldoc, xpath))) {
        objunlock(xmldoc);
        objunref(xmldoc);
        return NULL;
    }

    encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
    child = xmlNewDocNode(xmldoc->doc, NULL, (const xmlChar *)name, encval);
    ;
    xmlFree(encval);
    xmlAddChild(parent, child);

    if (attrkey && keyval) {
        encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)keyval);
        xmlSetProp(child, (const xmlChar *)attrkey, (const xmlChar *)encval);
        xmlFree(encval);
    }
    objunlock(xmldoc);

    if (!(newnode = xml_nodetohash(xmldoc, child, attrkey))) {

```

```

        objunref(xmldoc);
        return NULL;
    }

    objunref(xmldoc);
    return newnode;
}

```

### 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

<i>xmldoc</i>	Reference to XML document.
<i>xpath</i>	Path to add the node too.
<i>child</i>	XML node to append to path.

Definition at line 625 of file [libxml2.c](#).

References [xml\\_node::nodeptr](#), [objlock\(\)](#), [objref\(\)](#), [objunlock\(\)](#), and [objunref\(\)](#).

```

xmlNodePtr parent;
{
    xmlNodePtr parent;

    if (!objref(xmldoc)) {
        return;
    }

    objlock(xmldoc);
    if (!(parent = xml_getparent(xmldoc, xpath))) {
        objunlock(xmldoc);
        objunref(xmldoc);
    }

    xmlAddChild(parent, child->nodeptr);
    objunlock(xmldoc);
    objunref(xmldoc);
}

```

### 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\(\)](#).

```

{
    if (xml_has_init_parser) {
        objunref(xml_has_init_parser);
    }
}

```

### 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**

<i>xmldoc</i>	Reference to XML document.
<i>xpath</i>	Path to create.

Definition at line 507 of file `libxml2.c`.

References `xml_node::name`, `objlock()`, `objref()`, `objunlock()`, `objunref()`, and `xml_addnode()`.

```

{
    struct xml_node *nn;
    xmlXPathObjectPtr xpathObj;
    char *lpath, *tok, *save, *cpath, *dup;
    const char *root = (char *)xmldoc->root->name;
    int len;

    if (!objref(xmldoc)) {
        return;
    }

    if (!(dup = strdup(xpath))) {
        objunref(xmldoc);
        return;
    }

    len = strlen(xpath)+1;
    if (!(cpath = malloc(len))) {
        free(dup);
        objunref(xmldoc);
        return;
    }
    if (!(lpath = malloc(len))) {
        free(dup);
        free(cpath);
        objunref(xmldoc);
        return;
    }

    cpath[0] = '\0';
    lpath[0] = '\0';

#ifdef __WIN32__
    for (tok = strtok_r(dup, "/", &save); tok ; tok = strtok_r(NULL, "/", &
        save)) {
#else
    for (tok = strtok_s(dup, "/", &save); tok ; tok = strtok_s(NULL, "/", &
        save)) {
#endif
        strcat(cpath, "/");
        strcat(cpath, tok);
        if (!strcmp(tok, root)) {
            strcat(lpath, "/");
            strcat(lpath, tok);
            continue;
        }

        objlock(xmldoc);
        if (!(xpathObj = xmlXPathEvalExpression((const xmlChar *)cpath, xmldoc
            ->xpathCtx))) {
            objunlock(xmldoc);
            free(lpath);
            free(cpath);
            free(dup);
            objunref(xmldoc);
            return;
        }
        objunlock(xmldoc);

        if (xmlXPathNodeSetIsEmpty(xpathObj->nodelist)) {
            nn = xml_addnode(xmldoc, lpath, tok, NULL, NULL, NULL);
            objunref(nn);
        }

        xmlXPathFreeObject(xpathObj);
        strcat(lpath, "/");
        strcat(lpath, tok);
    }
}

```



```

    free(dup);
    free(lpath);
    free(cpath);
    objunref(xmldoc);
}

```

### 12.23.3.5 void xml\_delete ( struct xml\_node \* xnode )

Delete a node from document it is not unrefd and should be.

#### Parameters

<i>xnode</i>	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()`.

```

{
    objlock(xnode);
    xmlUnlinkNode(xnode->nodeptr);
    xmlFreeNode(xnode->nodeptr);
    xnode->nodeptr = NULL;
    objunlock(xnode);
}

```

### 12.23.3.6 void\* xml\_doctobuffer ( struct xml\_doc \* xmldoc )

Return a dump of a XML document.

The result can be accessed using `xml_getbuffer()`

#### Parameters

<i>xmldoc</i>	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()`.

```

{
    struct xml_buffer *xmlbuf;

    if (!(xmlbuf = objalloc(sizeof(*xmlbuf), xml_free_buffer))) {
        return NULL;
    }

    objlock(xmldoc);
    xmlDocDumpFormatMemory(xmldoc->doc, &xmlbuf->buffer, &xmlbuf->size, 1);
    objunlock(xmldoc);
    return xmlbuf;
}

```

### 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\(\)](#).

```

    {
        struct xml_buffer *xb = data;
        xmlFree(xb->buffer);
    }

```

**12.23.3.8 const char\* xml\_getattr ( struct xml\_node \* *xnode*, const char \* *attr* )**

Return value of attribute.

**Parameters**

<i>xnode</i>	XML node reference.
<i>attr</i>	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](#).

```

    {
        struct xml_attr *ainfo;

        if (!xnode) {
            return NULL;
        }

        if ((ainfo = bucket_list_find_key(xnode->attrs,
            attr))) {
            objunref(ainfo);
            return ainfo->value;
        } else {
            return NULL;
        }
    }

```

**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**

<i>buffer</i>	Reference to a xml_buffer struct.
---------------	-----------------------------------

Definition at line 712 of file [libxml2.c](#).

```

    {
        struct xml_buffer *xb = buffer;
    }

```

```

    if (!xb) {
        return NULL;
    }
    return (char *)xb->buffer;
}

```

### 12.23.3.10 struct xml\_node\* xml\_getfirstnode ( struct xml\_search \* xpsearch, void \*\* iter ) [read]

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 preferred.

#### Warning

This is not thread safe.

#### Parameters

<i>xpsearch</i>	XML xpath search to find first node.
<i>iter</i>	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`.

```

{
    struct xml_node_iter *newiter;
    struct xml_node *xn;

    if (!objref(xpsearch)) {
        return NULL;
    }

    if (iter) {
        newiter = objalloc(sizeof(*newiter), free_iter);
        objlock(xpsearch);
        newiter->cnt = xml_nodecount(xpsearch);
        objunlock(xpsearch);
        newiter->curpos = 0;
        newiter->xsearch = xpsearch;
        objref(newiter->xsearch);
        *iter = newiter;
    }

    xn = xml_gethash(xpsearch, 0, NULL);
    objunref(xpsearch);
    return xn;
}

```

### 12.23.3.11 struct xml\_node\* xml\_getnextnode ( void \* iter ) [read]

Return the next node.

## Parameters

<i>iter</i>	Iterator set in call to <code>xml_getfirstnode</code> .
-------------	---

## 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`.

```

{
    struct xml_node_iter *xi = iter;
    struct xml_node *xn;

    if (!objref(xi->xsearch)) {
        return NULL;
    }

    objlock(xi);
    xi->curpos++;
    if (xi->curpos >= xi->cnt) {
        objunlock(xi);
        objunref(xi->xsearch);
        return NULL;
    }
    xn = xml_gethash(xi->xsearch, xi->curpos, NULL);
    objunlock(xi);
    objunref(xi->xsearch);

    return xn;
}

```

#### 12.23.3.12 `struct xml_node* xml_getnode ( struct xml_search * xsearch, const char * key )` [read]

Return a node in the search matching key.

The key is matched against the index attribute supplied or the value of the node.

## Parameters

<i>xsearch</i>	Reference to xpath search.
<i>key</i>	Value to use to find node matched against 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`.

```

{
    if (!xsearch) {
        return NULL;
    }
    return bucket_list_find_key(xsearch->nodes, key);
}

```

#### 12.23.3.13 `struct bucket_list* xml_getnodes ( struct xml_search * xpsearch )` [read]

Return reference to bucket list containing nodes.

**Note**

use of this is preferred to [xml\\_getfirstnode\(\)](#) / [xml\\_getnextnode\(\)](#) if search order is not a issue.

**Parameters**

<i>xpsearch</i>	Reference to xpath search result returned by <a href="#">xml_xpath()</a> .
-----------------	--

**Returns**

Reference to bucket list containing nodes.

Definition at line [349](#) of file [libxml2.c](#).

References [xml\\_search::nodes](#), and [objref\(\)](#).

```

return (xpsearch && objref(xpsearch->nodes)) ? xpsearch->nodes
: NULL;
}

```

**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**

<i>xmldoc</i>	XML Document.
---------------	---------------

Definition at line [458](#) of file [libxml2.c](#).

```

if (xmldoc) {
    return (const char *)xmldoc->root->name;
}
return NULL;
}

```

**12.23.3.15 struct xml\_node\* xml\_getrootnode ( struct xml\_doc \* xmldoc ) [read]**

Return reference to the root node.

**Parameters**

<i>xmldoc</i>	XML Document to find root in.
---------------	-------------------------------

Definition at line [276](#) of file [libxml2.c](#).

References [objlock\(\)](#), and [objunlock\(\)](#).

```

struct xml_node *rn;
objlock(xmldoc);
rn = xml_nodetohash(xmldoc, xmldoc->root, NULL);
objunlock(xmldoc);
return rn;
}

```

### 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\(\)](#).

```

    {
    if (!xml_has_init_parser) {
        xml_has_init_parser = objalloc(0, free_parser);
        xmlInitParser();
        LIBXML_TEST_VERSION
        xmlKeepBlanksDefault(0);
        xmlLoadExtDtdDefaultValue = 1;
        xmlSubstituteEntitiesDefault(1);
    } else {
        objref(xml_has_init_parser);
    }
}

```

### 12.23.3.17 struct xml\_doc\* xml\_loadbuf ( const uint8\_t \* buffer, uint32\_t len, int validate ) [read]

Load a buffer into XML document returning reference.

#### Parameters

<i>buffer</i>	Buffer containing the XML.
<i>len</i>	Size of the buffer.
<i>validate</i>	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\(\)](#).

```

    {
    struct xml_doc *xmldata;
    int flags;

    xml_init();

    if (!(xmldata = objalloc(sizeof(*xmldata), free_xmldata))) {
        return NULL;
    }

    if (validate) {
        flags = XML_PARSE_DTDLOAD | XML_PARSE_DTDVALID;
    } else {
        flags = XML_PARSE_DTDVALID;
    }

    if (!(xmldata->doc = xmlReadMemory((const char *)buffer, len, NULL, NULL,
        flags))) {
        objunref(xmldata);
        return NULL;
    }
    return xml_setup_parse(xmldata, 0);
}

```

**12.23.3.18** `struct xml_doc* xml_loaddoc ( const char * docfile, int validate )` [read]

Load a XML file into XML document and return reference.

#### Parameters

<i>docfile</i>	Pathname to XML file.
<i>validate</i>	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()`.

```

{
    struct xml_doc *xmldata;

    xml_init();

    if (!(xmldata = objalloc(sizeof(*xmldata), free_xmldata))) {
        return NULL;
    }

    if (!(xmldata->doc = xmlParseFile(docfile))) {
        objunref(xmldata);
        return NULL;
    }

    return xml_setup_parse(xmldata, validate);
}

```

**12.23.3.19** `void xml_modify ( struct xml_doc * xmldoc, struct xml_node * xnode, const char * value )`

Modify a XML node.

#### Parameters

<i>xmldoc</i>	XML Document node belongs to
<i>xnode</i>	XML Node to modify.
<i>value</i>	Value to set.

Definition at line 469 of file `libxml2.c`.

References `ALLOC_CONST`, `xml_node::nodeptr`, `objlock()`, `objunlock()`, and `xml_node::value`.

```

{
    xmlChar *encval;
    xmlNodePtr node;

    objlock(xmldoc);
    node = xnode->nodeptr;
    encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
    xmlNodeSetContent(node, encval);
    xmlFree(encval);
    encval = xmlNodeListGetString(xmldoc->doc, node->xmlChildrenNode, 1);
    objunlock(xmldoc);

    if (xnode->value) {
        free((void*)xnode->value);
    }
    ALLOC_CONST(xnode->value, (const char *)encval);
    xmlFree(encval);
}

```

### 12.23.3.20 int xml\_nodecount ( struct xml\_search \* xsearch )

Return the number of nodes in the search path.

#### Parameters

<i>xsearch</i>	Reference to XML xpath search ( <a href="#">xml_xpath()</a> )
----------------	---

#### Returns

Number of of nodes.

Definition at line 413 of file [libxml2.c](#).

References [xml\\_search::xpathObj](#).

Referenced by [xml\\_getfirstnode\(\)](#).

```

{
    xmlNodeSetPtr nodeset;

    if (xsearch && xsearch->xpathObj && ((nodeset = xsearch->xpathObj
        ->nodesetval))) {
        return nodeset->nodeNr;
    } else {
        return 0;
    }
}

```

### 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

<i>xmldoc</i>	Reference to XML document to save.
<i>file</i>	Filename to write the XML document too.
<i>format</i>	Formating flag from libxml2.
<i>compress</i>	Compression level 0[none]-9.

Definition at line 769 of file [libxml2.c](#).

References [objlock\(\)](#), and [objunlock\(\)](#).

```

{
    objlock(xmldoc);
    xmlSetDocCompressMode(xmldoc->doc, compress);
    xmlSaveFormatFile(file, xmldoc->doc, format);
    xmlSetDocCompressMode(xmldoc->doc, 0);
    objunlock(xmldoc);
}

```

### 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

<i>xmldoc</i>	XML Document node belongs to
<i>xnode</i>	XML Node to modify.
<i>name</i>	Attribute to modify.
<i>value</i>	Value to set.



Definition at line 493 of file `libxml2.c`.

References `xml_node::nodeptr`, `objlock()`, and `objunlock()`.

```

{
    xmlChar *encval;

    objlock(xmlDoc);
    encval = xmlEncodeSpecialChars(xmlDoc->doc, (const xmlChar *)value);
    xmlSetProp(xnode->nodeptr, (const xmlChar *)name, (const xmlChar *)
        encval);
    objunlock(xmlDoc);
    xmlFree(encval);
}

```

### 12.23.3.23 void xml\_unlink ( struct xml\_node \* *xnode* )

Unlink a node from the document.

#### Parameters

<i>xnode</i>	Reference of node to unlink.
--------------	------------------------------

Definition at line 693 of file `libxml2.c`.

References `xml_node::nodeptr`, `objlock()`, and `objunlock()`.

```

{
    objlock(xnode);
    xmlUnlinkNode(xnode->nodeptr);
    objunlock(xnode);
}

```

### 12.23.3.24 struct xml\_search\* xml\_xpath ( struct xml\_doc \* *xmldata*, const char \* *xpath*, const char \* *attrkey* ) [read]

Return a reference to a xpath search result.

#### Parameters

<i>xmldata</i>	XML Document to search.
<i>xpath</i>	Xpath search to apply.
<i>attrkey</i>	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`.

```

{
    struct xml_search *xpsearch;

    if (!objref(xmldata) || !(xpsearch = objalloc(sizeof(*
        xpsearch), free_xmlsearch))) {
        return NULL;
    }

    objlock(xmldata);
    xpsearch->xmldoc = xmldata;
}

```

```
if (!(xpsearch->xpathObj = xmlXPathEvalExpression((const xmlChar *)
xpath, xmldata->xpathCtx))) {
    objunlock(xmldata);
    objunref(xpsearch);
    return NULL;
}

if (xmlXPathNodeSetIsEmpty(xpsearch->xpathObj->nodesetval)) {
    objunlock(xmldata);
    objunref(xpsearch);
    return NULL;
}
objunlock(xmldata);

if (!(xpsearch->nodes = xml_setnodes(xpsearch, attrkey))) {
    objunref(xpsearch);
    return NULL;
}
return xpsearch;
}
```

## 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 declaration 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](#) \*xslt\_doc, const char \*param, const char \*value)  
*Add a parameter to the XSLT document.*
- void [xslt\\_clearparam](#) (struct [xslt\\_doc](#) \*xslt\_doc)  
*Delete all parameters of a XSLT document.*
- void [xslt\\_apply](#) (struct [xml\\_doc](#) \*xmldoc, struct [xslt\\_doc](#) \*xslt\_doc, const char \*filename, int comp)  
*Apply XSLT document to a XML document.*
- void \* [xslt\\_apply\\_buffer](#) (struct [xml\\_doc](#) \*xmldoc, struct [xslt\\_doc](#) \*xslt\_doc)  
*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 declaration 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

<i>xsltdoc</i>	Reference to XSLT document.
<i>param</i>	Name of parameter.
<i>value</i>	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](#).

```

    {
        struct xslt_param *xparam;
        int size;

        if (!xsltdoc || !xsltdoc->params || !objref(xsltdoc) || !(
            xparam = objalloc(sizeof(*xparam), free_param))) {
            return;
        }

        size = strlen(value) + 3;
        ALLOC_CONST(xparam->name, param);
        xparam->value = malloc(size);
        snprintf((char *)xparam->value, size, "'%s'", value);
        objlock(xsltdoc);
        addtobucket(xsltdoc->params, xparam);
        objunlock(xsltdoc);
        objunref(xparam);
        objunref(xsltdoc);
    }

```

#### 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

<i>xmldoc</i>	Reference to XML document.
<i>xsltdoc</i>	Reference to XSLT document.
<i>filename</i>	File to write the result too.
<i>comp</i>	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\(\)](#).

```

    {
        const char **params = NULL;
        xmlDocPtr res;

        /* ref's xml/xslt locks xslt IF set*/
        if (!(params = xslt_params(xmldoc, xsltdoc))) {
            return;
        }

#ifdef __WIN32__
        touch(filename, 80, 80);
#else
        touch(filename);
#endif
        objlock(xmldoc);
        res = xsltApplyStylesheet(xsltdoc->doc, xmldoc->doc, params);
    }

```

```

xsltSaveResultToFilename(filename, res, xsltdoc->doc, comp);
objunlock(xmlDoc);
objunref(xmlDoc);
objunlock(xsltdoc);

free(params);
xmlFreeDoc(res);
xslt_clearparam(xsltdoc);
objunref(xsltdoc);
}

```

### 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

<i>xmlDoc</i>	Reference to XML document.
<i>xsltdoc</i>	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\(\)](#).

```

{
    struct xml_buffer *xmlbuf;
    const char **params;
    xmlDocPtr res;

    if (!(xmlbuf = objalloc(sizeof(*xmlbuf), xml_free_buffer))) {
        return NULL;
    }

    if (!(params = xslt_params(xmlDoc, xsltdoc))) {
        objunref(xmlbuf);
        return NULL;
    }

    objlock(xmlDoc);
    res = xsltApplyStylesheet(xsltdoc->doc, xmlDoc->doc, params);
    xsltSaveResultToString(&xmlbuf->buffer, &xmlbuf->size, res, xsltdoc->doc);
    objunlock(xmlDoc);
    objunref(xmlDoc);
    objunlock(xsltdoc);

    free(params);
    xmlFreeDoc(res);
    xslt_clearparam(xsltdoc);
    objunref(xsltdoc);

    return xmlbuf;
}

```

### 12.24.3.4 void xslt\_clearparam ( struct xslt\_doc \* xsltdoc )

Delete all parameters of a XSLT document.

#### Parameters

<i>xsltdoc</i>	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\(\)](#).

```

    {
        if (!xslt_doc || !xslt_doc->params) {
            return;
        }

        objlock(xslt_doc);
        objunref(xslt_doc->params);
        xslt_doc->params = create_bucketlist(0, xslt_hash);
        objunlock(xslt_doc);
    }

```

#### 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\(\)](#).

```

    {
        if (xslt_has_init_parser) {
            objunref(xslt_has_init_parser);
        }
    }

```

#### 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\(\)](#).

```

    {
        if (!xslt_has_init_parser) {
            xslt_has_init_parser=objalloc(0, free_parser);
        } else {
            objref(xslt_has_init_parser);
        }
    }

```

#### 12.24.3.7 struct xslt\_doc\* xslt\_open ( const char \* *xsltfile* ) [read]

Open a XSLT file returning reference to it.

##### Parameters

<i>xsltfile</i>	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\(\)](#).

```

{
    struct xslt_doc *xsltdoc;

    if (!(xsltdoc = objalloc(sizeof(*xsltdoc), free_xsltdoc))) {
        return NULL;
    }
    xslt_init();

    xsltdoc->doc = xsltParseStylesheetFile((const xmlChar *)xsltfile);
    xsltdoc->params = create_bucketlist(0, xslt_hash);
    return xsltdoc;
}
```

## 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 Authentication callback.*
- struct [curl\\_post](#)  
*HTTP post data structure.*
- struct [basic\\_auth](#)  
*Basic authentication structure.*
- struct [curlbuf](#)  
*Buffer containing the result of a curl transaction.*

### Typedefs

- typedef struct [curl\\_post](#) [curl\\_post](#)  
*Forward declaration of structure.*
- typedef struct [basic\\_auth](#) [basic\\_auth](#) (\*)(const char \*, const char \*, void \*)  
*Callback to set the authentication 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 *)` [read]

Callback to set the authentication ie on error 401.

#### Parameters

<i>user</i>	Initial username (currently set)
<i>passwd</i>	Initial password (currently set)
<i>data</i>	Reference to data passed.

#### Returns

New auth structure to re attempt authentication.

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

<i>clientp</i>	Reference to userdata supplied.
<i>dltotal</i>	Total download bytes.
<i>dlnow</i>	Current bytes downloaded.

<i>ultotal</i>	Total upload bytes.
<i>ulnow</i>	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

<i>data</i>	Reference to userdata supplied to <a href="#">curl_setprogress()</a> .
-------------	--

#### 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

<i>data</i>	Reference to userdata supplied.
<i>state</i>	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 )` [read]

Create a XML document from from buffer (application/xml)

#### Parameters

<i>cbuf</i>	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\(\)](#).

```

{
    struct xml_doc *xmldoc = NULL;

    if (cbuf && cbuf->c_type && !strcmp("application/xml", cbuf->c_type)) {
        curl_ungzip(cbuf);
        xmldoc = xml_loadbuf(cbuf->body, cbuf->bsize, 1);
    }
    return xmldoc;
}

```

### 12.25.3.2 struct curlbuf\* curl\_geturl ( const char \* *def\_url*, struct basic\_auth \* *bauth*, curl\_authcb *authcb*, void \* *auth\_data* ) [read]

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**

<i>def_url</i>	URL to fetch.
<i>bauth</i>	Basic auth structure to initilise auth.
<i>authcb</i>	Callback if authentication is required.
<i>auth_data</i>	Reference to userdata passed in auth callback.

**Returns**

CURL buffer structure.

Definition at line 276 of file [curl.c](#).

```

{
    return curl_sendurl(def_url, bauth, NULL, authcb, auth_data);
}

```

### 12.25.3.3 struct basic\_auth\* curl\_newauth ( const char \* *user*, const char \* *passwd* ) [read]

Create a new auth structure with initial vallues.

**Note**

if NULL is supplied its replaced with zero length string

**Parameters**

<i>user</i>	Optional initial username to set.
<i>passwd</i>	Optional initial password to set.

**Returns**

Reference to new authentication structure.

Definition at line 328 of file [curl.c](#).

References [objalloc\(\)](#), [basic\\_auth::passwd](#), and [basic\\_auth::user](#).

```

{
    struct basic_auth *bauth;

    if (!(bauth = (struct basic_auth *)objalloc(sizeof(*bauth)
        ), curl_freeauth))) {
        return NULL;
    }
    if (user) {
        bauth->user = strdup(user);
    } else {
        bauth->user = strdup("");
    }
    if (passwd) {
        bauth->passwd = strdup(passwd);
    } else {
        bauth->passwd = strdup("");
    }
    return bauth;
}

```

**12.25.3.4 struct curl\_post\* curl\_newpost ( void ) [read]**

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\(\)](#).

```

{
    struct curl_post *post;
    if (!(post = objalloc(sizeof(*post), free_post))) {
        return NULL;
    }
    post->first = NULL;
    post->last = NULL;
    return post;
}

```

**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**

<i>post</i>	Post structure created with <a href="#">curl_newpost()</a>
<i>name</i>	Name of the pair.
<i>value</i>	Value of the pair.

Definition at line 370 of file [curl.c](#).

References [curl\\_post::first](#), [curl\\_post::last](#), [objlock\(\)](#), and [objunlock\(\)](#).

```

{
    if (!name || !value) {

```

```

    return;
}
objlock(post);
curl_formadd(&post->first, &post->last,
    CURLFORM_COPYNAME, name,
    CURLFORM_COPYCONTENTS, value,
    CURLFORM_END);
objunlock(post);
}

```

**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 )` [read]

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

<i>def_url</i>	URL to fetch.
<i>bauth</i>	Basic auth structure to initilise auth.
<i>post</i>	Reference to curl post structure.
<i>authcb</i>	Callback if authentication is required.
<i>auth_data</i>	Reference to userdata passed in auth callback.

#### Returns

CURL buffer structure.

Definition at line 288 of file `curl.c`.

```

    {
return curl_sendurl(def_url, bauth, post, authcb, auth_data);
}

```

**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

<i>auth_cb</i>	Authentication call back.
<i>data</i>	Reference to userdata passed in callback.

Definition at line 470 of file `curl.c`.

References `objalloc()`, `objref()`, and `objunref()`.

```

    {
if (curlpassword) {
    objunref(curlpassword);
    curlpassword = NULL;
}

```

```

    }

    if (!(curlpassword = objjalloc(sizeof(*curlpassword),
        free_curlpassword))) {
        return;
    }

    curlpassword->authcb = auth_cb;
    if (data && objjref(data)) {
        curlpassword->data = data;
    }
}

```

#### 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

<i>cb</i>	CURL progress function callback.
<i>p_cb</i>	CURL progress control (pause) callback.
<i>d_cb</i>	CURL progress data allocation callback.
<i>data</i>	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 [objjalloc\(\)](#), [objjref\(\)](#), and [objjref\(\)](#).

```

{
    if (curlprogress) {
        objjref(curlprogress);
        curlprogress = NULL;
    }

    if (!(curlprogress = objjalloc(sizeof(*curlprogress), free_progress))
        ) {
        return;
    }
    curlprogress->cb = cb;
    curlprogress->d_cb = d_cb;
    curlprogress->p_cb = p_cb;
    if (data && objjref(data)) {
        curlprogress->data = data;
    }
}

```

#### 12.25.3.9 struct curlbuf\* curl.ungzip ( struct curlbuf \* cbuf ) [read]

If the buffer contains GZIP data uncompress it.

##### Parameters

<i>cbuf</i>	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\(\)](#).

```

{
    uint8_t *gzbuf;
    uint32_t len;

    if (is_gzip((uint8_t *)cbuf->body, cbuf->bsize) &&
        ((gzbuf = gzinflatebuf((uint8_t *)cbuf->body, cbuf
        ->bsize, &len))) {
        free(cbuf->body);
        cbuf->body = gzbuf;
        cbuf->bsize = len;
    }
    return cbuf;
}

```

**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\(\)](#).

```

{
    objunref(curl_isinit);
    curl_isinit = NULL;
}

```

**12.25.3.11 int curlinit ( void )**

Initilise the CURL library.

**Note**

Curl functions will initialize and unreference curl when done it is best the application hold a reference to benefit 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\(\)](#).

```

{
    if (curl_isinit) {
        return objref(curl_isinit);
    }

    if (!(curl_isinit = objalloc(sizeof(void *), curlfree))) {
        return 0;
    }

    objlock(curl_isinit);
    if (!(curl = curl_easy_init())) {
        objunlock(curl_isinit);
        objunref(curl_isinit);
        return 0;
    }

    curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0);
    curl_easy_setopt(curl, CURLOPT_NOSIGNAL, 1);
}

```

```

curl_easy_setopt(curl, CURLOPT_COOKIEFILE, "");

curl_easy_setopt(curl, CURLOPT_USERAGENT, "libcurl-agent/1.0 [Distro
Solutions]");

curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, bodytobuffer);
curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, headertobuffer);
objunlock(curl_isinit);
return 1;
}

```

### 12.25.3.12 char\* url\_escape ( char \* url )

Escape and return the url.

#### Parameters

<i>url</i>	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\(\)](#).

```

{
char *esc;
char *ret = NULL;

if (!curlinit()) {
    return NULL;
}

objlock(curl_isinit);
esc = curl_easy_escape(curl, url, 0);
if (esc) {
    ret = strdup(esc);
}
curl_free(esc);
objunlock(curl_isinit);
objunref(curl_isinit);
return ret;
}

```

### 12.25.3.13 char\* url\_unescape ( char \* url )

UN escape and return the url.

#### Parameters

<i>url</i>	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\(\)](#).

```

{
char *uesc;
char *ret = NULL;

```



```
if (!curlinit()) {
    return NULL;
}

objlock(curl_isinit);
uesc = curl_easy_unescape(curl, url, 0, 0);
if (uesc) {
    ret = strdup(uesc);
}
curl_free(uesc);
objunlock(curl_isinit);
objunref(curl_isinit);
return ret;
}
```

## 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

<i>buf_in</i>	Buffer to inflate.
<i>buf_size</i>	Size of buf_in buffer.
<i>len</i>	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\(\)](#).

```
z_stream zdat;
uint8_t *buf = NULL, *tmp;
int res;
```

```
{
```

```

zdat.opaque = NULL;
zdat.zalloc = NULL;
zdat.zfree = NULL;

zdat.next_in = buf_in;
zdat.avail_in = buf_size;
zdat.next_out = buf;
zdat.avail_out = 0;
zdat.total_out = 0;

if (inflateInit2(&zdat, 31)) {
    return NULL;
}

do {
    if (! (tmp = realloc(buf, zdat.total_out + (zdat.avail_in * 5) + 1))) {
        res = Z_MEM_ERROR;
        break;
    } else {
        buf = tmp;
    }
    buf[zdat.total_out] = '\0';
    zdat.next_out = &buf[zdat.total_out];
    zdat.avail_out += zdat.avail_in * 5;
} while ((res = inflate(&zdat, Z_NO_FLUSH)) == Z_OK);

if (res == Z_STREAM_END) {
    buf = realloc(buf, zdat.total_out);
    *len = zdat.total_out;
} else {
    free(buf);
    *len = 0;
    buf = NULL;
}
inflateEnd(&zdat);

return buf;
}

```

### 12.26.2.2 int is\_gzip ( uint8\_t \* buf, int buf\_size )

check a buffer if it contains gzip magic

#### Parameters

<i>buf</i>	buffer to check.
<i>buf_size</i>	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\(\)](#).

```

{
    if (buf_size < 4) {
        return 0;
    }
    if (memcmp(buf, gzipMagicBytes, 4)) {
        return 0;
    }
    return 1;
}

```

### 12.26.2.3 struct zobj\* zcompress ( uint8\_t \* buff, uint16\_t len, uint8\_t level ) [read]

Allocate a buffer and return it with compressed data.

## Parameters

<i>buff</i>	Buffer to compress.
<i>len</i>	Length of the buffer.
<i>level</i>	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](#).

```

{
    struct zobj *ret;

    if (!(ret = objalloc(sizeof(*ret), zobj_free))) {
        return (NULL);
    }

    ret->zlen = compressBound(len);
    ret->olen = len;

    if (!(ret->buff = malloc(ret->zlen))) {
        return (NULL);
    }
    compress2(ret->buff, (uLongf *)&ret->zlen, buff, len, level);

    return (ret);
}

```

#### 12.26.2.4 void `zuncompress ( struct zobj * buff, uint8_t * obuff )`

Uncompress zobj buffer to buffer.

## Parameters

<i>buff</i>	Compressed buffer to uncompress.
<i>obuff</i>	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](#).

```

{
    uLongf olen = buff->olen;

    if (!obuff) {
        return;
    }

    uncompress(obuff, &olen, buff->buff, buff->zlen);
}

```

## 12.27 Burtle Bob hash algorithim.

[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 jenkins(key, length, initval) hashlittle(key, length, (initval) ? initval : JHASH_INITVAL);`  
*Define jenkins 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 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 jenkins( key, length, initval ) hashlittle(key, length, (initval) ? initval : JHASH_INITVAL);`

Define jenkins 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](#).

```

uint32_t a,b,c;
union {
    const void *ptr;
    size_t i;
} u; /* to cast key to (size_t) happily */

/* Set up the internal state */
a = b = c = 0xdeadbeef + ((uint32_t)length) + initval;

u.ptr = key;
if (HASH_BIG_ENDIAN && ((u.i & 0x3) == 0)) {
    const uint32_t *k = (const uint32_t *)key;          /* read 32-bit

```



```

        chunks */
#ifdef VALGRIND
    const uint8_t *k8;
#endif

    /*----- all but last block: aligned reads and affect 32 bits of
    (a,b,c) */
    while (length > 12) {
        a += k[0];
        b += k[1];
        c += k[2];
        mix(a,b,c);
        length -= 12;
        k += 3;
    }

    /*----- handle the last (probably partial)
    block */
    /*
    * "k[2]<<8" actually reads beyond the end of the string, but
    * then shifts out the part it's not allowed to read. Because the
    * string is aligned, the illegal read is in the same word as the
    * rest of the string. Every machine with memory protection I've seen
    * does it on word boundaries, so is OK with this. But VALGRIND will
    * still catch it and complain. The masking trick does make the hash
    * noticeably faster for short strings (like English words).
    */
#ifdef VALGRIND

    switch(length) {
        case 12:
            c+=k[2];
            b+=k[1];
            a+=k[0];
            break;
        case 11:
            c+=k[2]&0xffffffff00;
            b+=k[1];
            a+=k[0];
            break;
        case 10:
            c+=k[2]&0xffff0000;
            b+=k[1];
            a+=k[0];
            break;
        case 9 :
            c+=k[2]&0xff000000;
            b+=k[1];
            a+=k[0];
            break;
        case 8 :
            b+=k[1];
            a+=k[0];
            break;
        case 7 :
            b+=k[1]&0xffffffff00;
            a+=k[0];
            break;
        case 6 :
            b+=k[1]&0xffff0000;
            a+=k[0];
            break;
        case 5 :
            b+=k[1]&0xff000000;
            a+=k[0];
            break;
        case 4 :
            a+=k[0];
            break;
        case 3 :
            a+=k[0]&0xffffffff00;
            break;
        case 2 :
            a+=k[0]&0xffff0000;
            break;
        case 1 :
            a+=k[0]&0xff000000;
            break;
        case 0 :
            return (c);          /* zero length strings require no
mixing */
    }

#else /* make valgrind happy */

    k8 = (const uint8_t *)k;
    switch(length) {          /* all the case statements fall
through */

```

```

        case 12:
            c+=k[2];
            b+=k[1];
            a+=k[0];
            break;
        case 11:
            c+=((uint32_t)k8[10])<<8; /* fall through */
        case 10:
            c+=((uint32_t)k8[9])<<16; /* fall through */
        case 9 :
            c+=((uint32_t)k8[8])<<24; /* fall through */
        case 8 :
            b+=k[1];
            a+=k[0];
            break;
        case 7 :
            b+=((uint32_t)k8[6])<<8; /* fall through */
        case 6 :
            b+=((uint32_t)k8[5])<<16; /* fall through */
        case 5 :
            b+=((uint32_t)k8[4])<<24; /* fall through */
        case 4 :
            a+=k[0];
            break;
        case 3 :
            a+=((uint32_t)k8[2])<<8; /* fall through */
        case 2 :
            a+=((uint32_t)k8[1])<<16; /* fall through */
        case 1 :
            a+=((uint32_t)k8[0])<<24;
            break;
        case 0 :
            return c;
    }

#ifdef /* !VALGRIND */

} else { /* need to read the key one byte at a time
*/
    const uint8_t *k = (const uint8_t *)key;

    /*----- all but the last block: affect some 32 bits of
(a,b,c) */
    while (length > 12) {
        a += ((uint32_t)k[0])<<24;
        a += ((uint32_t)k[1])<<16;
        a += ((uint32_t)k[2])<<8;
        a += ((uint32_t)k[3]);
        b += ((uint32_t)k[4])<<24;
        b += ((uint32_t)k[5])<<16;
        b += ((uint32_t)k[6])<<8;
        b += ((uint32_t)k[7]);
        c += ((uint32_t)k[8])<<24;
        c += ((uint32_t)k[9])<<16;
        c += ((uint32_t)k[10])<<8;
        c += ((uint32_t)k[11]);
        mix(a,b,c);
        length -= 12;
        k += 12;
    }

    /*----- last block: affect all 32 bits of
(c) */
    switch(length) { /* all the case statements fall
through */
        case 12:
            c+=k[11];
            /* no break */
        case 11:
            c+=((uint32_t)k[10])<<8;
            /* no break */
        case 10:
            c+=((uint32_t)k[9])<<16;
            /* no break */
        case 9 :
            c+=((uint32_t)k[8])<<24;
            /* no break */
        case 8 :
            b+=k[7];
            /* no break */
        case 7 :
            b+=((uint32_t)k[6])<<8;
            /* no break */
        case 6 :
            b+=((uint32_t)k[5])<<16;
            /* no break */
        case 5 :

```

```

        b+=((uint32_t)k[4])<<24;
        /* no break */
    case 4 :
        a+=k[3];
        /* no break */
    case 3 :
        a+=((uint32_t)k[2])<<8;
        /* no break */
    case 2 :
        a+=((uint32_t)k[1])<<16;
        /* no break */
    case 1 :
        a+=((uint32_t)k[0])<<24;
        break;
    case 0 :
        return (c);
    }
}

final(a,b,c);
return (c);
}

```

### 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<sup>32</sup> is acceptable. Do NOT use for cryptographic purposes.

-----

Definition at line 298 of file lookup3.c.

References [HASH\\_LITTLE\\_ENDIAN](#), and [mix](#).

```

uint32_t a,b,c;
/*
union {
    const void *ptr;
    size_t i;
} u;    /* needed for Mac Powerbook G4 */

/* Set up the internal state */
a = b = c = 0xdeadbeef + ((uint32_t)length) + initval;

u.ptr = key;
if (HASH_LITTLE_ENDIAN && ((u.i & 0x3) == 0)) {
    const uint32_t *k = (const uint32_t *)key;    /* read 32-bit
    chunks */
#ifdef VALGRIND
    const uint8_t *k8;
#endif
    /*----- all but last block: aligned reads and affect 32 bits of
    {
    /* internal state
    */

```

```

(a,b,c) */
while (length > 12) {
    a += k[0];
    b += k[1];
    c += k[2];
    mix(a,b,c);
    length -= 12;
    k += 3;
}

/*----- handle the last (probably partial)
block */
/*
 * "k[2]&0xffffffff" actually reads beyond the end of the string, but
 * then masks off the part it's not allowed to read. Because the
 * string is aligned, the masked-off tail is in the same word as the
 * rest of the string. Every machine with memory protection I've seen
 * does it on word boundaries, so is OK with this. But VALGRIND will
 * still catch it and complain. The masking trick does make the hash
 * noticeably faster for short strings (like English words).
 */
#ifdef VALGRIND

switch(length) {
    case 12:
        c+=k[2];
        b+=k[1];
        a+=k[0];
        break;
    case 11:
        c+=k[2]&0xffffffff;
        b+=k[1];
        a+=k[0];
        break;
    case 10:
        c+=k[2]&0xffff;
        b+=k[1];
        a+=k[0];
        break;
    case 9 :
        c+=k[2]&0xff;
        b+=k[1];
        a+=k[0];
        break;
    case 8 :
        b+=k[1];
        a+=k[0];
        break;
    case 7 :
        b+=k[1]&0xffffffff;
        a+=k[0];
        break;
    case 6 :
        b+=k[1]&0xffff;
        a+=k[0];
        break;
    case 5 :
        b+=k[1]&0xff;
        a+=k[0];
        break;
    case 4 :
        a+=k[0];
        break;
    case 3 :
        a+=k[0]&0xffffffff;
        break;
    case 2 :
        a+=k[0]&0xffff;
        break;
    case 1 :
        a+=k[0]&0xff;
        break;
    case 0 :
        return (c);          /* zero length strings require no
mixing */
}

#else /* make valgrind happy */

k8 = (const uint8_t *)k;
switch(length) {
    case 12:
        c+=k[2];
        b+=k[1];
        a+=k[0];
        break;
    case 11:

```

```

        c+=((uint32_t)k8[10])<<16; /* fall through */
    case 10:
        c+=((uint32_t)k8[9])<<8; /* fall through */
    case 9:
        c+=k8[8]; /* fall through */
    case 8:
        b+=k[1];
        a+=k[0];
        break;
    case 7:
        b+=((uint32_t)k8[6])<<16; /* fall through */
    case 6:
        b+=((uint32_t)k8[5])<<8; /* fall through */
    case 5:
        b+=k8[4]; /* fall through */
    case 4:
        a+=k[0];
        break;
    case 3:
        a+=((uint32_t)k8[2])<<16; /* fall through */
    case 2:
        a+=((uint32_t)k8[1])<<8; /* fall through */
    case 1:
        a+=k8[0];
        break;
    case 0:
        return c;
    }
#endif /* !valgrind */

} else
    if (HASH_LITTLE_ENDIAN && ((u.i & 0x1) == 0)) {
        const uint16_t *k = (const uint16_t *)key; /* read 16-bit
chunks */
        const uint8_t *k8;

        /*----- all but last block: aligned reads and different
mixing */
        while (length > 12) {
            a += k[0] + (((uint32_t)k[1])<<16);
            b += k[2] + (((uint32_t)k[3])<<16);
            c += k[4] + (((uint32_t)k[5])<<16);
            mix(a,b,c);
            length -= 12;
            k += 6;
        }

        /*----- handle the last (probably partial)
block */
        k8 = (const uint8_t *)k;
        switch(length) {
            case 12:
                c+=k[4]+(((uint32_t)k[5])<<16);
                b+=k[2]+(((uint32_t)k[3])<<16);
                a+=k[0]+(((uint32_t)k[1])<<16);
                break;
            case 11:
                c+=((uint32_t)k8[10])<<16; /* fall through */
                /* no break */
            case 10:
                c+=k[4];
                b+=k[2]+(((uint32_t)k[3])<<16);
                a+=k[0]+(((uint32_t)k[1])<<16);
                break;
            case 9:
                c+=k8[8]; /* fall through */
                /* no break */
            case 8:
                b+=k[2]+(((uint32_t)k[3])<<16);
                a+=k[0]+(((uint32_t)k[1])<<16);
                break;
            case 7:
                b+=((uint32_t)k8[6])<<16; /* fall through */
                /* no break */
            case 6:
                b+=k[2];
                a+=k[0]+(((uint32_t)k[1])<<16);
                break;
            case 5:
                b+=k8[4]; /* fall through */
                /* no break */
            case 4:
                a+=k[0]+(((uint32_t)k[1])<<16);
                break;
            case 3:
                a+=((uint32_t)k8[2])<<16; /* fall through */

```

```

        /* no break */
    case 2 :
        a+=k[0];
        break;
    case 1 :
        a+=k8[0];
        break;
    case 0 :
        return (c); /* zero length requires no
mixing */
    }

    } else { /* need to read the key one byte at a
time */
        const uint8_t *k = (const uint8_t *)key;

        /*----- all but the last block: affect some 32 bits of
(a,b,c) */
        while (length > 12) {
            a += k[0];
            a += ((uint32_t)k[1])<<8;
            a += ((uint32_t)k[2])<<16;
            a += ((uint32_t)k[3])<<24;
            b += k[4];
            b += ((uint32_t)k[5])<<8;
            b += ((uint32_t)k[6])<<16;
            b += ((uint32_t)k[7])<<24;
            c += k[8];
            c += ((uint32_t)k[9])<<8;
            c += ((uint32_t)k[10])<<16;
            c += ((uint32_t)k[11])<<24;
            mix(a,b,c);
            length -= 12;
            k += 12;
        }

        /*----- last block: affect all 32 bits
of (c) */
        switch(length) { /* all the case statements fall
through */
            case 12:
                c+=((uint32_t)k[11])<<24;
                /* no break */
            case 11:
                c+=((uint32_t)k[10])<<16;
                /* no break */
            case 10:
                c+=((uint32_t)k[9])<<8;
                /* no break */
            case 9 :
                c+=k[8];
                /* no break */
            case 8 :
                b+=((uint32_t)k[7])<<24;
                /* no break */
            case 7 :
                b+=((uint32_t)k[6])<<16;
                /* no break */
            case 6 :
                b+=((uint32_t)k[5])<<8;
                /* no break */
            case 5 :
                b+=k[4];
                /* no break */
            case 4 :
                a+=((uint32_t)k[3])<<24;
                /* no break */
            case 3 :
                a+=((uint32_t)k[2])<<16;
                /* no break */
            case 2 :
                a+=((uint32_t)k[1])<<8;
                /* no break */
            case 1 :
                a+=k[0];
                break;
            case 0 :
                return (c);
        }
    }

    final(a,b,c);
    return (c);
}

```

## 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](#).

```
uint32_t a,b,c; /* IN: secondary initval, OUT: secondary hash */
/* internal state */
union {
    const void *ptr;
    size_t i;
} u; /* needed for Mac Powerbook G4 */

/* Set up the internal state */
a = b = c = 0xdeadbeef + ((uint32_t)length) + *pc;
c += *pb;

u.ptr = key;
if (HASH_LITTLE_ENDIAN && ((u.i & 0x3) == 0)) {
    const uint32_t *k = (const uint32_t *)key; /* read 32-bit
    chunks */
#ifdef VALGRIND
    const uint8_t *k8;
#endif

    /*----- all but last block: aligned reads and affect 32 bits of
    (a,b,c) */
    while (length > 12) {
        a += k[0];
        b += k[1];
        c += k[2];
        mix(a,b,c);
        length -= 12;
        k += 3;
    }

    /*----- handle the last (probably partial)
    block */
    /*
    * "k[2]&0xffffffff" actually reads beyond the end of the string, but
    * then masks off the part it's not allowed to read. Because the
    * string is aligned, the masked-off tail is in the same word as the
    * rest of the string. Every machine with memory protection I've seen
    * does it on word boundaries, so is OK with this. But VALGRIND will
    * still catch it and complain. The masking trick does make the hash
    * noticeably faster for short strings (like English words).
    */
#ifdef VALGRIND
    switch(length) {
        case 12:
            c+=k[2];
            b+=k[1];
            a+=k[0];
            break;
        case 11:
            c+=k[2]&0xffffffff;
            b+=k[1];
            a+=k[0];
            break;
        case 10:
            c+=k[2]&0xffff;
            b+=k[1];
            a+=k[0];
            break;
        case 9 :
            c+=k[2]&0xff;
            b+=k[1];
            a+=k[0];
            break;
    }
#endif
```

```

        case 8 :
            b+=k[1];
            a+=k[0];
            break;
        case 7 :
            b+=k[1]&0xffffffff;
            a+=k[0];
            break;
        case 6 :
            b+=k[1]&0xffff;
            a+=k[0];
            break;
        case 5 :
            b+=k[1]&0xff;
            a+=k[0];
            break;
        case 4 :
            a+=k[0];
            break;
        case 3 :
            a+=k[0]&0xffffffff;
            break;
        case 2 :
            a+=k[0]&0xffff;
            break;
        case 1 :
            a+=k[0]&0xff;
            break;
        case 0 :
            *pc=c;
            *pb=b;
            return; /* zero length strings require no mixing */
    }

#else /* make valgrind happy */

    k8 = (const uint8_t *)k;
    switch(length) {
        case 12:
            c+=k[2];
            b+=k[1];
            a+=k[0];
            break;
        case 11:
            c+=((uint32_t)k8[10])<<16; /* fall through */
        case 10:
            c+=((uint32_t)k8[9])<<8; /* fall through */
        case 9 :
            c+=k8[8]; /* fall through */
        case 8 :
            b+=k[1];
            a+=k[0];
            break;
        case 7 :
            b+=((uint32_t)k8[6])<<16; /* fall through */
        case 6 :
            b+=((uint32_t)k8[5])<<8; /* fall through */
        case 5 :
            b+=k8[4]; /* fall through */
        case 4 :
            a+=k[0];
            break;
        case 3 :
            a+=((uint32_t)k8[2])<<16; /* fall through */
        case 2 :
            a+=((uint32_t)k8[1])<<8; /* fall through */
        case 1 :
            a+=k8[0];
            break;
        case 0 :
            *pc=c;
            *pb=b;
            return; /* zero length strings require no mixing */
    }

#endif /* !valgrind */

} else
    if (HASH_LITTLE_ENDIAN && ((u.i & 0x1) == 0)) {
        const uint16_t *k = (const uint16_t *)key; /* read 16-bit
chunks */
        const uint8_t *k8;

        /*----- all but last block: aligned reads and different
mixing */
        while (length > 12) {
            a += k[0] + (((uint32_t)k[1])<<16);

```



```

        b += k[2] + (((uint32_t)k[3])<<16);
        c += k[4] + (((uint32_t)k[5])<<16);
        mix(a,b,c);
        length -= 12;
        k += 6;
    }

    /*----- handle the last (probably partial)
    block */
    k8 = (const uint8_t *)k;
    switch(length) {
        case 12:
            c+=k[4]+(((uint32_t)k[5])<<16);
            b+=k[2]+(((uint32_t)k[3])<<16);
            a+=k[0]+(((uint32_t)k[1])<<16);
            break;
        case 11:
            c+=((uint32_t)k8[10])<<16;      /* fall through */
            /* no break */
        case 10:
            c+=k[4];
            b+=k[2]+(((uint32_t)k[3])<<16);
            a+=k[0]+(((uint32_t)k[1])<<16);
            break;
        case 9 :
            c+=k8[8];                      /* fall through */
            /* no break */
        case 8 :
            b+=k[2]+(((uint32_t)k[3])<<16);
            a+=k[0]+(((uint32_t)k[1])<<16);
            break;
        case 7 :
            b+=((uint32_t)k8[6])<<16;      /* fall through */
            /* no break */
        case 6 :
            b+=k[2];
            a+=k[0]+(((uint32_t)k[1])<<16);
            break;
        case 5 :
            b+=k8[4];                      /* fall through */
            /* no break */
        case 4 :
            a+=k[0]+(((uint32_t)k[1])<<16);
            break;
        case 3 :
            a+=((uint32_t)k8[2])<<16;      /* fall through */
            /* no break */
        case 2 :
            a+=k[0];
            break;
        case 1 :
            a+=k8[0];
            break;
        case 0 :
            *pc=c;
            *pb=b;
            return; /* zero length strings require no mixing */
    }

    } else {
        /* need to read the key one byte at a
        time */
        const uint8_t *k = (const uint8_t *)key;

        /*----- all but the last block: affect some 32 bits of
        (a,b,c) */
        while (length > 12) {
            a += k[0];
            a += ((uint32_t)k[1])<<8;
            a += ((uint32_t)k[2])<<16;
            a += ((uint32_t)k[3])<<24;
            b += k[4];
            b += ((uint32_t)k[5])<<8;
            b += ((uint32_t)k[6])<<16;
            b += ((uint32_t)k[7])<<24;
            c += k[8];
            c += ((uint32_t)k[9])<<8;
            c += ((uint32_t)k[10])<<16;
            c += ((uint32_t)k[11])<<24;
            mix(a,b,c);
            length -= 12;
            k += 12;
        }

        /*----- last block: affect all 32 bits
        of (c) */
        switch(length) {
            /* all the case statements fall
            through */

```

```

        case 12:
            c+=((uint32_t)k[11])<<24;
            /* no break */
        case 11:
            c+=((uint32_t)k[10])<<16;
            /* no break */
        case 10:
            c+=((uint32_t)k[9])<<8;
            /* no break */
        case 9 :
            c+=k[8];
            /* no break */
        case 8 :
            b+=((uint32_t)k[7])<<24;
            /* no break */
        case 7 :
            b+=((uint32_t)k[6])<<16;
            /* no break */
        case 6 :
            b+=((uint32_t)k[5])<<8;
            /* no break */
        case 5 :
            b+=k[4];
            /* no break */
        case 4 :
            a+=((uint32_t)k[3])<<24;
            /* no break */
        case 3 :
            a+=((uint32_t)k[2])<<16;
            /* no break */
        case 2 :
            a+=((uint32_t)k[1])<<8;
            /* no break */
        case 1 :
            a+=k[0];
            break;
        case 0 :
            *pc=c;
            *pb=b;
            return; /* zero length strings require no mixing */
    }
}

final(a,b,c);
*pc=c;
*pb=b;
}

```

#### 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](#).

```

        {          /* the previous hash, or an arbitrary
value */
uint32_t a,b,c;

/* Set up the internal state */
a = b = c = 0xdeadbeef + (((uint32_t)length)<<2) + initval;

/*----- handle most of the key
*/
while (length > 3) {

```

```

        a += k[0];
        b += k[1];
        c += k[2];
        mix(a,b,c);
        length -= 3;
        k += 3;
    }

    /*----- handle the last 3 uint32_t's
    */
    switch(length) {                /* all the case statements fall through
    */
        case 3 :
            c+=k[2];
            /* no break */
        case 2 :
            b+=k[1];
            /* no break */
        case 1 :
            a+=k[0];
            final(a,b,c);
            /* no break */
        case 0: /* case 0: nothing left to add */
            break;
    }
    /*----- report the result
    */
    return (c);
}

```

**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](#).

```

        {                /* IN: more seed OUT: secondary hash
        value */
        uint32_t a,b,c;

        /* Set up the internal state */
        a = b = c = 0xdeadbeef + ((uint32_t)(length<<2)) + *pc;
        c += *pb;

        /*----- handle most of the key
        */
        while (length > 3) {
            a += k[0];
            b += k[1];
            c += k[2];
            mix(a,b,c);
            length -= 3;
            k += 3;
        }

        /*----- handle the last 3 uint32_t's
        */
        switch(length) {                /* all the case statements fall through
        */
            case 3 :
                c+=k[2];
                /* no break */
            case 2 :
                b+=k[1];
                /* no break */
            case 1 :
                a+=k[0];
                final(a,b,c);
                /* no break */
            case 0: /* case 0: nothing left to add */

```

```
        break;
    }
    /*----- report the result
    */
    *pc=c;
    *pb=b;
}
```

## 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 declaration 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 declaration 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

<i>map</i>	Nat map structure to procees against.
<i>ipaddr</i>	Address to transform.
<i>out</i>	Set to non zero if ipaddr is internal and must be transformed to external.

Definition at line 62 of file [rfc6296.c](#).

References [natmap::adj](#), [natmap::adjo](#), [natmap::epre](#), [natmap::ipre](#), and [natmap::mask](#).

```

uint16_t *addr_16 = (uint16_t *)&ipaddr->s6_addr;
uint32_t calc;
uint8_t cnt, *prefix, bitlen, bytelen;
uint16_t adj;

prefix = (out) ? map->epre : map->ipre;
adj = (out) ? map->adjo : map->adj;

if ((bitlen = map->mask % 8)) {
    bytelen = (map->mask - bitlen) / 8;
    bytelen++;
} else {
    bytelen = map->mask / 8;
}

/*as per RFC we handle /48 and longer /48 changes are reflected in SN*/
if ((bytelen == 6) && (~addr_16[3]) && (!bitlen)) {
    memcpy(&ipaddr->s6_addr, prefix, bytelen);
    calc = ntohs(addr_16[3]) + adj;
    addr_16[3] = htons((calc & 0xFFFF) + (calc >> 16));
    if (!~addr_16[3]) {
        addr_16[3] = 0;
    }
} else if ((bytelen > 6) && (bytelen < 15)) {
    /* find first non 0xFFFF word in lower 64 bits*/
    for(cnt = ((bytelen-1) >> 1) + 1; cnt < 8; cnt++) {
        if (!~addr_16[cnt]) {
            continue;
        }
        if (bitlen) {
            ipaddr->s6_addr[bytelen-1] = prefix[bytelen-1] | (ipaddr->
s6_addr[bytelen-1] & ((1 << (8 - bitlen)) - 1));
        } else {
            ipaddr->s6_addr[bytelen-1] = prefix[bytelen-1];
        }
        memcpy(&ipaddr->s6_addr, prefix, bytelen - 1);
        calc = ntohs(addr_16[cnt]) + adj;
        addr_16[cnt] = htons((calc & 0xFFFF) + (calc >> 16));
        if (!~addr_16[cnt]) {
            addr_16[cnt] = 0;
        }
        break;
    }
}
}
}

```

### 12.28.3.2 int rfc6296\_map\_add ( char \* *intaddr*, char \* *extaddr* )

Calculate and add a NAT map.

#### Parameters

<i>intaddr</i>	Internal prefix/subnet.
<i>extaddr</i>	External prefix/subnet.

Definition at line 111 of file [rfc6296.c](#).

References [addtobucket\(\)](#), [natmap::adj](#), [natmap::adjo](#), [checksum\(\)](#), [create\\_bucketlist\(\)](#), [natmap::epre](#), [natmap::ipre](#), [natmap::mask](#), [objalloc\(\)](#), and [objunref\(\)](#).

```

{
    struct natmap *map;
    uint16_t emask, imask, isum, esum, bytelen, bitlen;
    char inip[43], expip[43], *tmp2;
    struct in6_addr i6addr;
    uint32_t adj;

    strncpy(inip, intaddr, 43);
    if ((tmp2 = rindex(inip, '/')) {
        tmp2[0] = '\0';
        tmp2++;
        imask = atoi(tmp2);
    }
}

```

```

    } else {
        return (-1);
    }

    strncpy(exip, extaddr, 43);
    if ((tmp2 = rindex(exip, '/')) {
        tmp2[0] = '\0';
        tmp2++;
        emask = atoi(tmp2);
    } else {
        return (-1);
    }

    map = objalloc(sizeof(*map), NULL);
    map->mask = (emask > imask) ? emask : imask;

    /*rfc says we must zero extend this is what we do here looking at each
    supplied len*/
    /*external range*/
    inet_pton(AF_INET6, exip, &i6addr);
    if ((bitlen = emask % 8)) {
        bytelen = (emask - bitlen) / 8;
        i6addr.s6_addr[bytelen] &= ~(1 << (8 - bitlen)) - 1);
        bytelen++;
    } else {
        bytelen = emask / 8;
    }
    memcpy(map->epre, &i6addr.s6_addr, bytelen);

    /*internal range*/
    inet_pton(AF_INET6, inip, &i6addr);
    if ((bitlen = imask % 8)) {
        bytelen = (imask - bitlen) / 8;
        i6addr.s6_addr[bytelen] &= ~(1 << (8 - bitlen)) - 1);
        bytelen++;
    } else {
        bytelen = imask / 8;
    }
    memcpy(map->ipre, &i6addr.s6_addr, bytelen);

    /*calculate the adjustments from checksums of prefixes*/
    if ((bitlen = map->mask % 8)) {
        bytelen = (map->mask - bitlen) / 8;
        bytelen++;
    } else {
        bytelen = map->mask / 8;
    }
    esum = ntohs(checksum(map->epre, bytelen));
    isum = ntohs(checksum(map->ipre, bytelen));

    /*outgoing transform*/
    adj = esum - isum;
    adj = (adj & 0xFFFF) + (adj >> 16);
    map->adjo = (uint16_t)adj;

    /*incoming transform*/
    adj = isum - esum;
    adj = (adj & 0xFFFF) + (adj >> 16);
    map->adji = (uint16_t)adj;

    if (!nptv6tbl && !(nptv6tbl = create_bucketlist(5,
        nptv6_hash))) {
        objunref(map);
        return (-1);
    }
    addtobucket(nptv6tbl, map);
    objunref(map);

    return (0);
}

```

### 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

<i>callback</i>	Bucket list callback.
<i>internal</i>	Ip addr passed as data to the callback.

Definition at line 197 of file [rfc6296.c](#).

References [bucketlist\\_callback\(\)](#), and [objunref\(\)](#).

```
/*find and run map*/
bucketlist_callback(nptv6tbl, callback, internal);
objunref(nptv6tbl);
}
```



## 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* ) [read]

Return interface info for a specified interface.

#### Parameters

<i>iface</i>	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\(\)](#).

```

PIP_ADAPTER_ADDRESSES ainfo = NULL, cinfo;
PIP_ADAPTER_UNICAST_ADDRESS pUnicast;
struct sockaddr_storage *ss;
char tmpfn[NI_MAXHOST];

```

```

char host4[NI_MAXHOST];
char host6[NI_MAXHOST];
int score4 = 0, score6 = 0, nscore;
struct ifinfo *ifinf = NULL;

if (!(ainfo = get_adaptorinfo(15000, 3))) {
    return NULL;
}

for(cinfo = ainfo; cinfo; cinfo = cinfo->Next) {
    if (strcmp(cinfo->AdapterName, iface)) {
        continue;
    }

    if (!(ifinf = (struct ifinfo*)objalloc(sizeof(*ifinf),
free_ifinfo))) {
        return NULL;
    }

    ifinf->idx = (int)cinfo->IfIndex;

    if (cinfo->PhysicalAddressLength == 6) {
        unsigned int i;
        char tmp[4];
        char tmp2[18] = "";
        for (i = 0; i < cinfo->PhysicalAddressLength; i++) {
            if (i == (cinfo->PhysicalAddressLength - 1)) {
                sprintf(tmp, "%.2X", (int)cinfo->PhysicalAddress[i]);
            } else {
                sprintf(tmp, "%.2X:", (int)cinfo->PhysicalAddress[i]);
            }
            strcat(tmp2, tmp);
        }
        ifinf->ifaddr = strdup(tmp2);
    } else {
        ifinf->ifaddr = NULL;
    }

    for (pUnicast = cinfo->FirstUnicastAddress; pUnicast; pUnicast =
pUnicast->Next) {
        ss = (struct sockaddr_storage*)pUnicast->Address.lpSockaddr;
        switch(ss->ss_family) {
            case AF_INET:
                nscore = score_ipv4((struct sockaddr_in*)ss,
tmpn, NI_MAXHOST);
                if (score4 < nscore) {
                    score4 = nscore;
                    strcpy(host4, tmpn);
                }
                break;
            case AF_INET6:
                nscore = score_ipv6((struct sockaddr_in6*)ss,
tmpn, NI_MAXHOST);
                if (score6 < nscore) {
                    score6 = nscore;
                    strcpy(host6, tmpn);
                }
                break;
        }
    }
    ifinf->ipv4addr = (strlenzero(host4)) ? NULL : strdup
(host4);
    ifinf->ipv6addr = (strlenzero(host6)) ? NULL : strdup
(host6);
    break;
}

if (ainfo) {
    free(ainfo);
}

return ifinf;
}

```

### 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

<i>af</i>	Address family only AF_INET or AF_INET6 are supported.
<i>src</i>	A pointer to in_addr or in6_addr.
<i>dest</i>	A buffer to place the IP address in.
<i>size</i>	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\(\)](#).

```

{
    union sockstruct sa;
    int res = 0;
    char serv[NI_MAXSERV];

    memset(&sa, 0, sizeof(sa));
    sa.ss.ss_family = af;

    switch(af) {
        case AF_INET:
            memcpy(&sa.sa4.sin_addr, src, sizeof(struct in_addr));
            res = getnameinfo(&sa.sa, sizeof(struct sockaddr_in), dest, size,
                serv, NI_MAXSERV, NI_NUMERICHOST | NI_NUMERICSERV);
            break;
        case AF_INET6:
            memcpy(&sa.sa6.sin6_addr, src, sizeof(struct in6_addr));
            res = getnameinfo(&sa.sa, sizeof(struct sockaddr_in6), dest, size,
                , serv, NI_MAXSERV, NI_NUMERICHOST | NI_NUMERICSERV);
            break;
    }
    return (!res) ? dest : NULL;
}

```

## 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 declaration of structure.*

### Enumerations

- enum [NF\\_CTRACK\\_FLAGS](#) { [NFCTTRACK\\_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 declaration 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:

**NFCTTRACK\_DONE**

Definition at line 44 of file [nf\\_ctrack.c](#).

```

NFCTTRACK_DONE    = 1 << 0
};

```

### 12.32.4 Function Documentation

#### 12.32.4.1 struct nf\_conntrack\* nf\_ctrack\_buildct ( uint8\_t \* pkt ) [read]

Definition at line 97 of file [nf\\_ctrack.c](#).

Referenced by [nf\\_ctrack\\_delete\(\)](#), and [nf\\_ctrack\\_nat\(\)](#).

```

{
    struct nf_conntrack *ct;
    struct iphdr *ip = (struct iphdr *)pkt;
    union l4hdr *l4 = (union l4hdr *) (pkt + (ip->ihl * 4));

    if (!(ct = nfct_new())) {
        return (NULL);
    };

    /*Build tuple*/
    nfct_set_attr_u8(ct, ATTR_L3PROTO, PF_INET);
    nfct_set_attr_u32(ct, ATTR_IPV4_SRC, ip->saddr);
    nfct_set_attr_u32(ct, ATTR_IPV4_DST, ip->daddr);
    nfct_set_attr_u8(ct, ATTR_L4PROTO, ip->protocol);
    switch(ip->protocol) {
        case IPPROTO_TCP:
            nfct_set_attr_u16(ct, ATTR_PORT_SRC, l4->tcp.source);
            nfct_set_attr_u16(ct, ATTR_PORT_DST, l4->tcp.dest);
            break;
        case IPPROTO_UDP:
            nfct_set_attr_u16(ct, ATTR_PORT_SRC, l4->udp.source);
            nfct_set_attr_u16(ct, ATTR_PORT_DST, l4->udp.dest);
            break;
        case IPPROTO_ICMP:
            nfct_set_attr_u8(ct, ATTR_ICMP_TYPE, l4->icmp.type);
            nfct_set_attr_u8(ct, ATTR_ICMP_CODE, l4->icmp.code);
            nfct_set_attr_u16(ct, ATTR_ICMP_ID, l4->icmp.un.echo.id);
            /* no break */
        default:
            break;
    };

    return (ct);
}

```

#### 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\(\)](#).

```

{
    if (ctrack) {

```

```

    objunref(ctrack);
}
ctrack = NULL;
}

```

#### 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\(\)](#).

```

{
    struct nf_conntrack *ct;
    uint8_t unref = 0;
    uint8_t ret = 0;

    if (!ctrack) {
        if (nf_ctrack_init()) {
            return (-1);
        }
        unref = 1;
    }

    ct = nf_ctrack_buildct(pkt);
    objlock(ctrack);
    if (nfct_query(ctrack->nfct, NFCT_Q_DESTROY, ct) < 0) {
        ret = -1;
    }
    objunlock(ctrack);
    nfct_destroy(ct);

    if (unref) {
        nf_ctrack_close();
    }

    return (ret);
}

```

#### 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\(\)](#).

```

{
    uint32_t family = PF_INET;
    uint8_t unref = 0;

    if (!ctrack) {
        if (nf_ctrack_init()) {
            return;
        }
        unref = 1;
    }

    objlock(ctrack);
    nfct_callback_register(ctrack->nfct, NFCT_T_ALL, nfct_cb, NULL);
    nfct_query(ctrack->nfct, NFCT_Q_DUMP, &family);
    nfct_callback_unregister(ctrack->nfct);
    objunlock(ctrack);

    if (unref) {
        nf_ctrack_close();
    }
}

```

#### 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](#).



```

    {
        if (nfct) {
            setflag(nfct, NFCTTRACK_DONE);
        }
        objunref(nfct);
    }

```

#### 12.32.4.6 uint8\_t nf\_cttrack\_init ( void )

Definition at line 90 of file [nf\\_cttrack.c](#).

Referenced by [nf\\_cttrack\\_delete\(\)](#), [nf\\_cttrack\\_dump\(\)](#), and [nf\\_cttrack\\_nat\(\)](#).

```

    {
        if (!ctrack && !(ctrack = nf_cttrack_alloc(CONNTRACK, 0))) {
            return (-1);
        }
        return (0);
    }

```

#### 12.32.4.7 uint8\_t nf\_cttrack\_nat ( uint8\_t \* pkt, uint32\_t addr, uint16\_t port, uint8\_t dnat )

Definition at line 160 of file [nf\\_cttrack.c](#).

References [nf\\_cttrack\\_buildct\(\)](#), [nf\\_cttrack\\_close\(\)](#), [nf\\_cttrack\\_init\(\)](#), [objlock\(\)](#), and [objunlock\(\)](#).

```

    {
        struct iphdr *ip = (struct iphdr *)pkt;
        struct nf_conntrack *ct;
        uint8_t unref = 0;
        uint8_t ret = 0;

        if (!ctrack) {
            if (nf_cttrack_init()) {
                return (-1);
            }
            unref = 1;
        }

        ct = nf_cttrack_buildct(pkt);
        nfct_setobjopt(ct, NFCT_SOPT_SETUP_REPLY);

        nfct_set_attr_u32(ct, ATTR_TIMEOUT, 120);
        nfct_set_attr_u32(ct, (dnat ? ATTR_DNAT_IPV4 : ATTR_SNAT_IPV4, addr);

        switch(ip->protocol) {
            case IPPROTO_TCP:
                nfct_set_attr_u8(ct, ATTR_TCP_STATE, TCP_CONNTRACK_ESTABLISHED);
                /* no break */
            case IPPROTO_UDP:
                if (port) {
                    nfct_set_attr_u16(ct, (dnat ? ATTR_DNAT_PORT : ATTR_SNAT_PORT,
                    port);
                }
                break;
        }

        objlock(ctrack);
        if (nfct_query(ctrack->nfct, NFCT_Q_CREATE_UPDATE, ct) < 0) {
            ret = -1;
        }
        objunlock(ctrack);
        nfct_destroy(ct);

        if (unref) {
            nf_cttrack_close();
        }

        return (ret);
    }

```

#### 12.32.4.8 struct nfct\_struct\* nf\_ctrack\_trace( void ) [read]

Definition at line 261 of file [nf\\_ctrack.c](#).

References [framework\\_mkthread\(\)](#), [objunref\(\)](#), and [THREAD\\_OPTION\\_RETURN](#).

```

{
    struct nfct_struct *nfct;
    void *thr;

    if (!(nfct = nf_ctrack_alloc(CONNTRACK, NFCT_ALL_CT_GROUPS))) {
        return (NULL);
    }

    if (!(thr = framework_mkthread(nf_ctrack_trace_th, NULL,
        NULL, nfct, THREAD_OPTION_RETURN))) {
        objunref(nfct);
        return NULL;
    }
    objunref(thr);

    return (nfct);
}
```

## 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 declaration of structure.*
- typedef struct [nfq\\_data](#) [nfq\\_data](#)  
*Forward declaration of structure.*
- typedef struct [nfqnl\\_msg\\_packet\\_hdr](#) [nfqnl\\_msg\\_packet\\_hdr](#)  
*Forward declaration 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 declaration of structure.

Definition at line 201 of file [dtsapp.h](#).

### 12.33.2.2 typedef struct nfq\_queue nfq\_queue

Forward declaration 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 declaration 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](#).

```

    {
        NFQUEUE_DONE = 1 << 0
    };

```

## 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 ) [read]

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](#).

```

    {
        struct nfq_queue *nfq_q;

        if (!(nfq_q = objalloc(sizeof(*nfq_q), nfqueue_close_q))) {
            return (NULL);
        }

        objlock(nfqueues);
        if (!(nfqueues && (nfq_q->nfq = bucket_list_find_key
            (nfqueues->queues, &pf))) &&
            !(nfq_q->nfq || (nfq_q->nfq = nfqueue_init(pf)))) {
            objunlock(nfqueues);
            objunref(nfq_q);
            return (NULL);
        }
        objunlock(nfqueues);

        if (!(nfq_q->qh = nfq_create_queue(nfq_q->nfq->h, num, &
            nfqueue_callback, nfq_q))) {
            objunref(nfq_q);
            return (NULL);
        }

        if (cb) {

```

```

        nfq_q->cb = cb;
    }

    if (data) {
        nfq_q->data = data;
    }

    nfq_set_mode(nfq_q->qh, mode, range);

    return (nfq_q);
}

```

#### 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()`.

```

{
    struct iphdr *ip = (struct iphdr *)pkt;
    char *tmp = buff;
    uint32_t id, mark, ifi;
    uint16_t tlen, left = len;
    char saddr[INET_ADDRSTRLEN], daddr[INET_ADDRSTRLEN];

    if (ph) {
        id = ntohl(ph->packet_id);
        snprintf(tmp, left, "hw_protocol=0x%04x hook=%u id=%u ",
                 ntohs(ph->hw_protocol), ph->hook, id);
        tlen = strlen(tmp);
        tmp += tlen;
        left -= tlen;
    }

    if ((mark = nfq_get_nfmark(tb))) {
        snprintf(tmp, left, "mark=%u ", mark);
        tlen = strlen(tmp);
        tmp += tlen;
        left -= tlen;
    }

    if ((ifi = nfq_get_indev(tb))) {
        snprintf(tmp, left, "indev=%u ", ifi);
        tlen = strlen(tmp);
        tmp += tlen;
        left -= tlen;
    }

    if ((ifi = nfq_get_outdev(tb))) {
        snprintf(tmp, left, "outdev=%u ", ifi);
        tlen = strlen(tmp);
        tmp += tlen;
        left -= tlen;
    }

    if (pkt && (ip->version == 4)) {
        union l4hdr *l4 = (union l4hdr *) (pkt + (ip->ihl*4));

        inet_ntop(AF_INET, &ip->saddr, saddr, INET_ADDRSTRLEN);
        inet_ntop(AF_INET, &ip->daddr, daddr, INET_ADDRSTRLEN);

        snprintf(tmp, left, "src=%s dst=%s proto=%i ", saddr, daddr, ip->
                 protocol);
        tlen = strlen(tmp);
        tmp += tlen;
        left -= tlen;

        switch(ip->protocol) {
            case IPPROTO_TCP:
                snprintf(tmp, left, "sport=%i dport=%i ", ntohs(l4->tcp.source),
                         ntohs(l4->tcp.dest));
                break;
            case IPPROTO_UDP:
                snprintf(tmp, left, "sport=%i dport=%i ", ntohs(l4->udp.source),
                         ntohs(l4->udp.dest));
                break;
            case IPPROTO_ICMP:
                snprintf(tmp, left, "type=%i code=%i id=%i ", l4->icmp.type, l4->
                         icmp.code, ntohs(l4->icmp.un.echo.id));
                break;
        }
    }
}

```

```
        break;
    }
    tlen = strlen(tmp);
    tmp += tlen;
    left -= tlen;
}
return (len - left);
}
```

## Chapter 13

# Data Structure Documentation

### 13.1 basic\_auth Struct Reference

Basic authentication structure.

```
#include <dtsapp.h>
```

#### Data Fields

- const char \* [user](#)  
*Username.*
- const char \* [passwd](#)  
*Password.*

#### 13.1.1 Detailed Description

Basic authentication 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  $2^n$  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  $2^{\text{bucketbits}}$ .

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_htppost * first`  
*First item in the list.*
- `struct curl_htppost * 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_htppost* 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_htppost* 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\(\)](#), [dtls\\_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 prioritised by Routed/Reserved/Zeroconf.*
- const char \* [ipv6addr](#)  
*IPv6 address prioritised 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 prioritised 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 prioritised 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 nlmsg\_hdr [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](#)  
*Distinguished 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

Distinguished 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 attribute.

```
#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 attribute.

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 [\\_dtsldapsearch\(\)](#), [ldap\\_connect\(\)](#), [ldap\\_doaddd\(\)](#), [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 [\\_dtsldapsearch\(\)](#), and [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::ctrlsp

LDAP control.

Definition at line 60 of file [openldap.c](#).

Referenced by [\\_dtsldapsearch\(\)](#), [ldap\\_connect\(\)](#), [ldap\\_doaddd\(\)](#), [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 [\\_dtsldapsearch\(\)](#), and [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 distinguished name.*
- `const char * dnufn`  
*LDAP user format distinguished 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 distinguished 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 distinguished 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 [\\_dtsldapsearch\(\)](#), and [ldap\\_unref\\_entry\(\)](#).

**13.19.2.7** `struct ldap_entry* ldap_entry::prev`

Previous entry.

Definition at line 771 of file [dtsapp.h](#).

Referenced by [\\_dtsldapsearch\(\)](#).

**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`  
*Distinguished name.*
- `struct bucket_list * bl [3]`  
*Bucket list containing 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 containing 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`

Distinguished 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 distinguished 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 distinguished 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 [\\_dtsldapsearch\(\)](#), and [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 [\\_dtsldapsearch\(\)](#), [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 [\\_dtsldapsearch\(\)](#), [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`  
*Distinguished 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`

Distinguished 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](#) [nfqueue::cb](#)

Definition at line [57](#) of file [nf\\_queue.c](#).

Referenced by [nfqueue\\_attach\(\)](#).

#### 13.27.2.2 [void\\*](#) [nfqueue::data](#)

Definition at line [58](#) of file [nf\\_queue.c](#).

Referenced by [nfqueue\\_attach\(\)](#).

#### 13.27.2.3 [struct nfq\\_struct\\*](#) [nfqueue::nfq](#)

Definition at line [55](#) of file [nf\\_queue.c](#).

Referenced by [nfqueue\\_attach\(\)](#).

#### 13.27.2.4 [uint16\\_t](#) [nfqueue::num](#)

Definition at line [59](#) of file [nf\\_queue.c](#).

#### 13.27.2.5 [struct nfq\\_q\\_handle\\*](#) [nfqueue::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 struct 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](#)]  
*Authentication 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]

Authentication 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 connexions.*

### 13.32.1 Detailed Description

Radius Server.

define a server with host auth/acct port and secret create "connexions" 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 connexions.

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 referenced objects been handled as referenced objects.*

- uint32\_t [cnt](#)

*Reference count the object 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 object 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 referenced 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 authentication.

### 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 authentication.

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 [dtls\\_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\(\)](#), [dtls\\_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\(\)](#), [dtls\\_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)  
*Based 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 daemomize 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 initilization 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

<a href="#">sock</a>	Reference to new socket
<a href="#">data</a>	Reference to data supplied on thread start

#### Examples:

[socket.c](#).

Definition at line 36 of file [socket.c](#).

Referenced by [socktest\(\)](#).

```

}
{

```

#### 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 thread callbacks and use of socketread / socketwrite is all that is required.

#### Parameters

<i>sock</i>	Reference to socket data is available on.
<i>data</i>	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\(\)](#).

```

{
    char buff[128];

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

```

#### 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\(\)](#).

```

    ://www.distrotech.co.za", 2013, "/var/run/sockettest",
    FRAMEWORK_FLAG_DAEMONLOCK, NULL) {

    if (argc < 3) {
#ifdef __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);
#ifdef __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");
    }
}

```

#### 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 around `recvfrom` and `socketwrite_d` is a wrapper around `sendto` this is important when dealing with un encrypted UDP sessions where the socket needs `sendto` addresss to send data too.

##### Parameters

<i>sock</i>	Reference to socket data is available on.
<i>data</i>	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\(\)](#).

```

{
    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);
    }
}

```

#### 14.10.2.5 void socktest ( const char \* ipaddr, int tcp, int ssl )

Based 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

<i>ipaddr</i>	As supplied on the command line
<i>tcp</i>	Set to non zero if using TCP.
<i>ssl</i>	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\(\)](#).

```

{
    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);
}

```

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

<i>socket</i>	File name to create server on and connect too.
<i>protocol</i>	This is either SOCK_STREAM or SOCK_DGRAM and are similar to TCP/UDP respectively.

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\(\)](#).

```

{
    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);
}

```

## 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
00131 #ifndef __WIN32
00132 void unixsockettest(const char *socket, int protocol) {
00133     char *buff = "client 1";
00134     char *buff2 = "client 2";
00135     struct fwsocket *client, *client2, *server;
00136
00137     server = unixsocket_server(socket, protocol, S_IXUSR |
S_IWGRP | S_IRGRP | S_IXGRP | S_IWOTH | S_IROTH | S_IXOTH, server_func,
NULL);
00138     sleep(1); /*wait for socket*/
00139     client = unixsocket_client(socket, protocol, client_func
, NULL);
00140     client2 = unixsocket_client(socket, protocol, client_func
, NULL);
00141
00142     socketwrite_d(client, buff, strlen(buff)+1, NULL);
00143     socketwrite_d(client2, buff2, strlen(buff2)+1, NULL);
00144
00145     sleep(5);
00146
00147     close_socket(client);
00148     close_socket(client2);
00149     close_socket(server);
00150 }
00151 #endif
00152
00153 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/socketest",
FRAMEWORK_FLAG_DAEMONLOCK, NULL) {
00154
00155     if (argc < 3) {
00156 #ifndef __WIN32
00157         printf("Requires arguments %s [tcp|tls|udp|dtls|unix_d|unix_s]
[ipaddr|socket]\n", argv[0]);
00158     #else
00159         printf("Requires arguments %s [tcp|tls|udp|dtls] ipaddr\n", argv[0]);
00160     #endif
00161     }
00162     return (-1);
00163 }
00164
00165 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](#) \* [socketconnect](#) (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](#) \* [udpbinding](#) (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 support encrypted sockets a ssldata reference 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@distrotech.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
00097 , sock);
00098         }
00099         objunref(sock->parent);
00100     }
00101
00102     /*looks like the server is shut down*/
00103     if (sock->children) {
00104         objunref(sock->children);
00105     }
00106
00107     if (sock->sock >= 0) {
00108         close(sock->sock);
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         {
00204             break;
00205         }
00206         objunref(sock);
00207         sock = NULL;
00208     }
00209
00210     if (!sock || !rp) {
00211         if (sock) {
00212             objunref(sock);
00213         }
00214         freeaddrinfo(result);
00215
00216         return (NULL);
00217     }
00218
00219     if (ctype) {
00220         sock->flags |= SOCK_FLAG_BIND;
00221         memcpy(&sock->addr.ss, rp->ai_addr, sizeof(sock->addr.ss));
00222         switch(sock->type) {
00223             case SOCK_STREAM:
00224             case SOCK_SEQPACKET:
00225                 listen(sock->sock, backlog);
00226                 /* no break */
00227             default:
00228                 break;
00229         }
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 interrupt continue or timed out*/
00362 #ifdef __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                 {
00370                     dtlshandltimeout(sock);
00371                 }
00372                 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             dtls_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 *srcip;
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*)&t1l
, 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
*)&t1l, 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 categories 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 Wrapper 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@distrotech.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 = jenkins(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 = jenkins(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
00077 extern void unrefconfigfiles(void) {
00078     if (configfiles) {
00079         objunref(configfiles);
00080     }
00081 }
00082
00083 static void free_config_entry(void *data) {
00084     struct config_entry *entry = data;
00085
00086     if (entry->item) {
00087         free((void *)entry->item);
00088     }
00089     if (entry->value) {
00090         free((void *)entry->value);
00091     }
00092 }
00093
00094 static void add_conf_entry(struct config_category *category,
00095     const char *item, const char *value) {
00096     struct config_entry *newentry;
00097
00098     if (!category || !category->entries || !(newentry = objalloc
00099     (sizeof(*newentry), free_config_entry))) {
00100         return;
00101     }
00102     ALLOC_CONST(newentry->item, item);
00103     ALLOC_CONST(newentry->value, value);
00104     addtobucket(category->entries, newentry);
00105     objunref(newentry);
00106 }
00107
00108 static void free_config_category(void *data) {
00109     struct config_category *cat = data;
00110
00111     if (cat->name) {
00112         free((void *)cat->name);
00113     }
00114     if (cat->entries) {
00115         objunref(cat->entries);
00116     }
00117 }
00118
00119 static struct config_category *create_conf_category(const char *
00120     name) {
00121     struct config_category *newcat;
00122
00123     if (!(newcat = objalloc(sizeof(*newcat), free_config_category))) {
00124         return (NULL);
00125     }
00126     ALLOC_CONST(newcat->name, name);
00127     newcat->entries = create_bucketlist(5, hash_cats);
00128
00129     return (newcat);
00130 }
00131
00132 static void free_config_file(void *data) {
00133     struct config_file *file = data;
00134
00135     if (file->filename) {
00136         free((void *)file->filename);
00137     }
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 Authentication 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 ) {
00049     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 ) {
00053         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 *
00061 userdata) {
00062     size_t bufsize = size * nmemb;
00062     struct curlbuf *mem = (struct curlbuf *)userdata;
00063
00064     if (!(mem->header = realloc(mem->header, mem->hsize +
00065     bufsize + 1))) {
00066         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 : "
00211", (auth) ? auth->passwd : "", auth_data)))) {
00212                         snprintf(userpass, 63, "%s:%s", auth->user, auth->
00213passwd);
00214                         curl_easy_setopt(curl, CURLOPT_USERPWD, userpass);
00215                         emptybuffer(writebuf);
00216                     } else {
00217                         i=3;
00218                     }
00219
00220                     if (curlprogress && curlprogress->p_cb) {
00221                         curlprogress->p_cb(p_data, 0);
00222                     }
00223                     break;
00224                 /*not found*/
00225                 case 300:
00226                     i=3;
00227                     break;
00228                 /*redirect*/
00229                 case 301:
00230                     curl_easy_getinfo(curl, CURLINFO_REDIRECT_URL, &url);
00231                     curl_easy_setopt(curl, CURLOPT_URL, url);
00232                     emptybuffer(writebuf);
00233                     i--;
00234                     break;
00235                 /*ok*/
00236                 case 200:
00237                     curl_easy_getinfo(curl, CURLINFO_CONTENT_TYPE, &writebuf->
00238c_type);
00239                     break;
00240             }
00241         }
00242         i++;
00243     } while ((res != 200) && (i < 3));
00244
00245     /* curl_easy_getinfo(curl, CURLINFO_COOKIELIST, &cookies);
00246     for(nc = cookies; nc; nc=nc->next) {
00247         printf("%s\n", nc->data);
00248     } */
00249
00250     if (!bauth) {
00251         objunref(auth);
00252     }
00253
00254     if (post) {
00255         objunlock(post);
00256         objunref(post);
00257     }
00258
00259     if (curlprogress && curlprogress->p_cb) {
00260         curlprogress->p_cb(p_data, -1);
00261     }
00262
00263     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 ) {
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  {
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(
00095          dir, user, group)) {
00096  #endif
00097          return 0;
00098      }
00099      return -1;
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 distinguished name linked list.*
- struct [ldap\\_attrval](#)  
*LDAP attribute value.*
- struct [ldap\\_attr](#)  
*LDAP attribute.*
- struct [ldap\\_entry](#)  
*LDAP entry.*
- struct [ldap\\_results](#)  
*LDAP results.*
- struct [basic\\_auth](#)  
*Basic authentication structure.*
- struct [curlbuf](#)  
*Buffer containing the result of a curl transaction.*

### Macros

- #define [RAD\\_AUTH\\_HDR\\_LEN](#) 20  
*Authentication 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 initialization 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 declaration of structure.*
- typedef struct [natmap](#) [natmap](#)
  - Forward declaration of structure.*
- typedef struct [radius\\_packet](#) [radius\\_packet](#)
  - Forward declaration of structure.*
- typedef struct [nfq\\_queue](#) [nfq\\_queue](#)
  - Forward declaration of structure.*
- typedef struct [nfq\\_data](#) [nfq\\_data](#)
  - Forward declaration of structure.*
- typedef struct [nfct\\_struct](#) [nfct\\_struct](#)
  - Forward declaration of structure.*

- typedef struct [nfqnl\\_msg\\_packet\\_hdr](#) [nfqnl\\_msg\\_packet\\_hdr](#)  
*Forward declaration 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 \*)  
*Callback used when processing config files.*
- typedef void(\* [config\\_catcb](#) )(struct [bucket\\_list](#) \*, const char \*)  
*Callback used when processing a category.*
- typedef void(\* [config\\_entrycb](#) )(const char \*, const char \*)  
*Callback used when processing an 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 declaration of structure.*
- typedef struct [xml\\_search](#) [xml\\_search](#)  
*Forward declaration of structure.*
- typedef struct [xml\\_doc](#) [xml\\_doc](#)  
*Forward declaration of structure.*
- typedef struct [xslt\\_doc](#) [xslt\\_doc](#)  
*Forward declaration of structure.*
- typedef struct [ldap\\_conn](#) [ldap\\_conn](#)  
*Forward declaration of structure.*
- typedef struct [ldap\\_modify](#) [ldap\\_modify](#)  
*Forward declaration of structure.*
- typedef struct [ldap\\_add](#) [ldap\\_add](#)  
*Forward declaration of structure.*
- typedef struct [curl\\_post](#) [curl\\_post](#)  
*Forward declaration of structure.*
- typedef struct [basic\\_auth](#) (\*(\* [curl\\_authcb](#) )(const char \*, const char \*, void \*))  
*Callback to set the authentication 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 controlling 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 \*progrname, char \*name, char \*email, char \*web, int year, char \*runfile, int flags, `sys sighandler` 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 beginning 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 address.*
- int [score\\_ipv6](#) (struct sockaddr\_in6 \*sa6, char \*ipaddr, int iplen)  
*Return a score for a IPv6 address.*
- int [inet\\_lookup](#) (int family, const char \*host, void \*addr, socklen\_t len)  
*Perform DNS lookup on a host/ip return 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 checksum of a IPv4 ICMP packet.*
- void [ipv4checksum](#) (uint8\_t \*pkt)

- Set the checksum of a IPv4 Packet.*
- int [packetchecksumv4](#) (uint8\_t \*pkt)
- Update the checksum of a IPv4 packet.*
- int [packetchecksumv6](#) (uint8\_t \*pkt)
- Prototype to check checksum 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 [cidrcont](#) (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 for 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)
- Randomly assign a SSM Multicast address.*
- void [mcast6\\_ip](#) (struct in6\_addr \*addr)
- Randomly 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\\_contrack](#) \* [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 [closenetwork](#) (void)  
*Close netlink socket on application termination.*
- const char \* [get\\_ifipaddr](#) (const char \*iface, int family)  
*Find best IP adress for a interface.*
- void [addraddtrint](#) (struct [radius\\_packet](#) \*packet, char type, unsigned int val)  
*Add a integer attribute too the packet.*
- void [addraddtrip](#) (struct [radius\\_packet](#) \*packet, char type, char \*ipaddr)  
*Add a integer attribute too the packet.*
- void [addraddtrstr](#) (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 [tisaccept](#) (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 Wrapper 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 reference.*
- 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 requiring a distinguished 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\\_rematr](#) (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 wrapper with support for GET/POST, authentication and progress indication.  
 File utilities as a wrapper around fstat.  
 IP 4/6 Utilities for calculating / checking subnets and checksumming 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,
00033  * 2009 The FreeRADIUS Server Project
00034  */
00035
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     struct sockaddr_un un;
00085 #endif
00086 #endif
00087
00088     struct sockaddr_in sa4;
00089     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_catchb)(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 *progrname;
00338     int year;
00340     int flock;
00342     struct sigaction *sa;
00345     syssighandler sig_handler;
00348     int flags;
00349 };
00350
00351 void framework_mkcore(char *progrname, 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 burtlebob*/
00397 extern uint32_t hashlitle(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 shasum(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 shasum2(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 shalcmp(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 shalhmac(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 *
xmldoc, 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,
    const char *passwd, const char *authzid);
00795 extern 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);
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_remattrib(struct ldap_conn *ldap,
      const char *dn, const char *attr);
00816 extern int ldap_mod_delattrib(struct ldap_conn *ldap,
      const char *dn, const char *attr, const char *value);
00817 extern int ldap_mod_addattrib(struct ldap_conn *ldap,
      const char *dn, const char *attr, const char *value);
00818 extern int ldap_mod_repattrib(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 around 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 precedence 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 RFC/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 kernel 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 address.*
- int [score\\_ipv6](#) (struct sockaddr\_in6 \*sa6, char \*ipaddr, int iplen)  
*Return a score for a IPv6 address.*
- const char \* [get\\_ifipaddr](#) (const char \*iface, int family)  
*Find best IP adress for a interface.*

### 14.22.1 Detailed Description

Wrapper around 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 nlmsgghdr n;
00089     struct ifinfomsg i;
00091     char buf[1024];
00092 };
00093
00095 struct ipaddr_req {
00097     struct nlmsgghdr 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(
00116         nlh, 0))) {
00117         if (nlh) {
00118             objunref(nlh);
00119         }
00120         return (NULL);
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.lmsg_len = NLMMSG_LENGTH(sizeof(struct ifinfomsg));
00181     req->n.lmsg_type = RTM_DELLINK;
00182     req->n.lmsg_flags = NLM_F_REQUEST;
00183
00184     /*config base/dev/mac*/
00185     req->i.ifindex = 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.msg_len = NLMMSG_LENGTH(sizeof(struct ifinfomsg));
00239     req->n.msg_type = RTM_NEWLINK;
00240     req->n.msg_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 = NLMMSG_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 = NLMMSG_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 *)NLMMSG_TAIL(&req->n) - (char *)data;
00257     linkinfo->rta_len = (char *)NLMMSG_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.msg_len = NLMMSG_LENGTH(sizeof(struct ifinfomsg));
00313     req->n.msg_type = RTM_NEWLINK;
00314     req->n.msg_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.nmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00367     req->n.nmsg_type = RTM_NEWLINK;
00368     req->n.nmsg_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.nmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
00401     req->n.nmsg_type = RTM_NEWLINK;
00402     req->n.nmsg_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 {
00423     struct iplink_req *req;
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.nlmmsg_len = NLMMSG_LENGTH(sizeof(struct ifinfomsg));
00434     req->n.nlmmsg_type = RTM_NEWLINK;
00435     req->n.nlmmsg_flags = NLM_F_REQUEST;
00436     req->i.ifindex = 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     /* set the MAC address*/
00520     if (hwaddr) {
00521         set_interface_addr(ifindex, hwaddr);
00522     }
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.nlmmsg_len = NLMMSG_LENGTH(sizeof(struct ifaddrmsg));
00632     req->n.nlmmsg_type = RTM_NEWADDR;
00633     req->n.nlmmsg_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
assignment*/
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     shalsum2(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) ||
00797             (ifa->ifa_addr->sa_family != family)) {
00798             continue;
00799         }
00800
00801         /* Match aliases not vlans*/
00802         if ((strlen(ifa->ifa_name) > iflen) && (ifa->ifa_name[iflen] != ':')) {
00803             continue;
00804         }
00805
00806         switch (ifa->ifa_addr->sa_family) {
00807             case AF_INET:
00808                 /* Find best ip address for a interface lowest priority is
00809                    given to zeroconf then reserved ip's
00810                    * finally find hte ip with shortest subnet bits.*/
00811                 ipv4addr = (struct sockaddr_in*)ifa->ifa_netmask;
00812                 match = ntohl(~ipv4addr->sin_addr.s_addr);
00813
00814                 nscore = score_ipv4((struct sockaddr_in*)ifa->
00815                     ifa_addr, tmp, NI_MAXHOST);
00816
00817                 /* match score and subnet*/
00818                 if ((nscore > score) || ((nscore == score) && (match > subnet)
00819 )) {
00820                     score = nscore;
00821                     subnet = match;
00822                     strncpy(host, tmp, NI_MAXHOST);
00823                 }
00824                 break;

```

```

00821         case AF_INET6:
00822             nscore = score_ipv6((struct sockaddr_in6*)ifa->
00823             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/iptutil.c File Reference

IPv4 And IPv6 Utilities.

```

#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 checksum of a IPv4 ICMP packet.*
- void [ipv4checksum](#) (uint8\_t \*pkt)  
*Set the checksum of a IPv4 Packet.*
- int [packetchecksumv4](#) (uint8\_t \*pkt)  
*Update the checksum of a IPv4 packet.*

- int [packetchecksumv6](#) (uint8\_t \*pkt)  
*Prototype to check checksum 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 for 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)  
*Randomly assign a SSM Multicast address.  
param addr Ip address structure to fill out.*
- void [mcast4\\_ip](#) (struct in\_addr \*addr)  
*Randomly 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 return 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 (!(0x00000000 ^ 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_ip4(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 reference.*
- 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 = jenkinshash(hashkey, strlen(hashkey), 0);
00101     } else {
00102         ret = jenkinshash(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 = jenkinshash(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->xmldata = 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     if (!objref(xmldoc)) {
00515         return;
00516     }
00517
00518     if (!(dup = strdup(xpath))) {
00519         objunref(xmldoc);
00520         return;
00521     }
00522
00523     len = strlen(xpath)+1;
00524     if (!(cpath = malloc(len))) {
00525         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     cpath[0] = '\0';
00537     lpath[0] = '\0';
00538 #ifndef __WIN32__
00539     for (tok = strtok_r(dup, "/", &save); tok ; tok = strtok_r(NULL, "/", &
00540 save)) {
00541 #else
00542     for (tok = strtok_s(dup, "/", &save); tok ; tok = strtok_s(NULL, "/", &
00543 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         objlock(xmldoc);
00553         if (!(xpathObj = xmlXPathEvalExpression((const xmlChar *)cpath, xmldoc
00554 ->xpathCtx))) {
00555             objunlock(xmldoc);
00556             free(lpath);
00557             free(cpath);
00558             free(dup);
00559             objunref(xmldoc);
00560             return;
00561         }
00562         objunlock(xmldoc);
00563         if (xmlXPathNodeSetIsEmpty(xpathObj->nodesetval)) {
00564             nn = xml_addnode(xmldoc, lpath, tok, NULL, NULL, NULL);
00565             objunref(nn);
00566         }
00567         xmlXPathFreeObject(xpathObj);
00568         strcat(lpath, "/");
00569         strcat(lpath, tok);
00570     }
00571     free(dup);
00572     free(lpath);
00573     free(cpath);
00574     objunref(xmldoc);
00575 }
00576
00577 static xmlNodePtr xml_getparent(struct xml_doc *xmldoc, const char *
00578 xpath) {
00579     xmlXPathObjectPtr xpathObj;
00580     xmlNodePtr parent = NULL;
00581     xmlNodeSetPtr nodes;
00582     int i, cnt;
00583     if (!(xpathObj = xmlXPathEvalExpression((const xmlChar *)xpath, xmldoc->
00584 xpathCtx))) {
00585         return NULL;
00586     }
00587     if (xmlXPathNodeSetIsEmpty(xpathObj->nodesetval)) {
00588         return NULL;
00589     }
00590     if (!(nodes = xpathObj->nodesetval)) {
00591         xmlXPathFreeObject(xpathObj);
00592         return NULL;
00593     }
00594     cnt = nodes->nodeNr;
00595     for (i=cnt - 1; i >= 0; i--) {
00596         if (nodes->nodeTab[i]->type == XML_ELEMENT_NODE) {
00597             parent=nodes->nodeTab[i];
00598             nodes->nodeTab[i] = NULL;
00599             break;
00600         }
00601     }
00602     return parent;
00603 }

```

```

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,
                                const char *attrkey, const char *keyval) {
00652     struct xml_node *newnode;
00653     xmlNodePtr parent;
00654     xmlNodePtr child;
00655     xmlChar *encval;
00656
00657     if (!objref(xmldoc)) {
00658         return NULL;
00659     }
00660
00661     objlock(xmldoc);
00662     if (!(parent = xml_getparent(xmldoc, xpath))) {
00663         objunlock(xmldoc);
00664         objunref(xmldoc);
00665         return NULL;
00666     }
00667
00668     encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)value);
00669     child = xmlNewDocNode(xmldoc->doc, NULL, (const xmlChar *)name, encval);
00670     xmlFree(encval);
00671     xmlAddChild(parent, child);
00672
00673     if (attrkey && keyval) {
00674         encval = xmlEncodeSpecialChars(xmldoc->doc, (const xmlChar *)keyval);
00675         xmlSetProp(child, (const xmlChar *)attrkey, (const xmlChar *)encval);
00676         xmlFree(encval);
00677     }
00678     objunlock(xmldoc);
00679
00680     if (!(newnode = xml_nodetohash(xmldoc, child, attrkey))) {
00681         objunref(xmldoc);
00682         return NULL;
00683     }
00684
00685     objunref(xmldoc);
00686
00687     return newnode;
00688 }
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 = jenkinshash(hashkey, strlen(hashkey), 0);
00058     } else {
00059         ret = jenkinshash(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/xsldoc and locks xsldoc*/
00127 static const char **xslt_params(struct xml_doc *xmldoc, struct xslt_doc
    *xsldoc) {
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(xsldoc)) {
00138         objunref(xmldoc);
00139         return NULL;
00140     }
00141
00142     objlock(xsldoc);
00143     if (!(params = malloc(sizeof(void *) * (bucket_list_cnt(
xslt_doc->params)*2 + 2)))) {
00144         objunlock(xsldoc);
00145         objunref(xsldoc);
00146         objunref(xmldoc);
00147         return NULL;
00148     }
00149
00150     bloop = init_bucket_loop(xsldoc->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
00162 extern void xslt_apply(struct xml_doc *xmldoc, struct xslt_doc
    *xsldoc, const char *filename, int comp) {
00163     const char **params = NULL;
00164     xmlDocPtr res;
00165
00166     /* ref's xml/xslt locks xslt IF set*/
00167     if (!(params = xslt_params(xmldoc, xsldoc))) {
00168         return;
00169     }
00170
00171     #ifndef __WIN32__
00172     touch(filename, 80, 80);
00173     #else
00174     touch(filename);
00175     #endif
00176
00177     objlock(xmldoc);
00178     res = xsltApplyStylesheet(xsldoc->doc, xmldoc->doc, params);
00179     xsltSaveResultToFile(filename, res, xsldoc->doc, comp);
00180     objunlock(xmldoc);
00181     objunref(xmldoc);
00182     objunlock(xsldoc);
00183
00184     free(params);
00185     xmlFreeDoc(res);
00186     xslt_clearparam(xsldoc);
00187     objunref(xsldoc);
00188 }
00189
00190 extern void *xslt_apply_buffer(struct xml_doc *xmldoc,
    struct xslt_doc *xsldoc) {
00191     struct xml_buffer *xmlbuf;
00192     const char **params;
00193     xmlDocPtr res;
00194
00195     if (!(xmlbuf = objalloc(sizeof(*xmlbuf), xml_free_buffer
    ))) {
00196         return NULL;
00197     }
00198
00199     if (!(params = xslt_params(xmldoc, xsldoc))) {
00200         objunref(xmlbuf);
00201         return NULL;
00202     }
00203
00204     objlock(xmldoc);
00205     res = xsltApplyStylesheet(xsldoc->doc, xmldoc->doc, params);
00206     xsltSaveResultToString(&xmlbuf->buffer, &xmlbuf->size, res, xsldoc->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 #ifndef 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
00184     size_t          length,      /* the length of the key, in
00185     uint32_ts *     uint32_ts */
00186     uint32_t        initval) {   /* the previous hash, or an arbitrary
00187     value */
00188     uint32_t a,b,c;
00189
00188     /* Set up the internal state */
00189     a = b = c = 0xdeadbeef + ((uint32_t)length)<<2 + initval;
00190
00191     /*----- handle most of the key
00192     */
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     */
00203     switch(length) {
00204         /* all the case statements fall through
00205         */
00206         case 3 :
00207             c+=k[2];
00208             /* no break */
00209         case 2 :
00210             b+=k[1];
00211             /* no break */
00212         case 1 :
00213             a+=k[0];
00214             final(a,b,c);
00215             /* no break */
00216         case 0: /* case 0: nothing left to add */
00217             break;
00218     }
00219     /*----- report the result
00220     */
00221     return (c);
00222 }
00223
00224 void hashword2 (
00225     const uint32_t *k,          /* the key, an array of uint32_t
00226     values */
00227     size_t length,             /* the length of the key, in
00228     uint32_ts */
00229     uint32_t *pc,              /* IN: seed OUT: primary hash
00230     value */
00231     uint32_t *pb) {            /* IN: more seed OUT: secondary hash
00232     value */
00233     uint32_t a,b,c;
00234
00235     /* Set up the internal state */
00236     a = b = c = 0xdeadbeef + ((uint32_t)(length<<2)) + *pc;
00237     c += *pb;
00238
00239     /*----- handle most of the key
00240     */
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     */
00252     switch(length) {
00253         /* all the case statements fall through
00254         */
00255         case 3 :
00256             c+=k[2];
00257             /* no break */
00258         case 2 :
00259             b+=k[1];
00260             /* no break */
00261         case 1 :
00262             a+=k[0];
00263             final(a,b,c);
00264             /* no break */
00265         case 0: /* case 0: nothing left to add */
00266             break;
00267     }
00268     /*----- report the result
00269     */
00270     *pc=c;
00271     *pb=b;
00272 }
00273
00274 uint32_t hashlittle( const void *key, size_t length, uint32_t initval
00275 ) {
00276     uint32_t a,b,c;          /* internal state
00277     */
00278     union {
00279         const void *ptr;
00280         size_t i;
00281     } 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]&0xffffffff" 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          * noticeably faster for short strings (like English words).
00333          */
00334 #ifndef VALGRIND
00335         switch(length) {
00336             case 12:
00337                 c+=k[2];
00338                 b+=k[1];
00339                 a+=k[0];
00340                 break;
00341             case 11:
00342                 c+=k[2]&0xffffffff;
00343                 b+=k[1];
00344                 a+=k[0];
00345                 break;
00346             case 10:
00347                 c+=k[2]&0xffff;
00348                 b+=k[1];
00349                 a+=k[0];
00350                 break;
00351             case 9 :
00352                 c+=k[2]&0xff;
00353                 b+=k[1];
00354                 a+=k[0];
00355                 break;
00356             case 8 :
00357                 b+=k[1];
00358                 a+=k[0];
00359                 break;
00360             case 7 :
00361                 b+=k[1]&0xffffffff;
00362                 a+=k[0];
00363                 break;
00364             case 6 :
00365                 b+=k[1]&0xffff;
00366                 a+=k[0];
00367                 break;
00368             case 5 :
00369                 b+=k[1]&0xff;
00370                 a+=k[0];
00371                 break;
00372             case 4 :
00373                 a+=k[0];
00374                 break;
00375             case 3 :
00376                 a+=k[0]&0xffffffff;
00377                 break;
00378             case 2 :
00379                 a+=k[0]&0xffff;
00380                 break;
00381             case 1 :
00382                 a+=k[0]&0xff;
00383                 break;
00384             case 0 :
00385                 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
00433         chunks */
00434
00435         const uint8_t *k8;
00436
00437         /*----- all but last block: aligned reads and different
00438         mixing */
00439         while (length > 12) {
00440             a += k[0] + (((uint32_t)k[1])<<16);
00441             b += k[2] + (((uint32_t)k[3])<<16);
00442             c += k[4] + (((uint32_t)k[5])<<16);
00443             mix(a,b,c);
00444             length -= 12;
00445             k += 6;
00446         }
00447
00448         /*----- handle the last (probably partial)
00449         block */
00450         k8 = (const uint8_t *)k;
00451         switch(length) {
00452         case 12:
00453             c+=k[4]+(((uint32_t)k[5])<<16);
00454             b+=k[2]+(((uint32_t)k[3])<<16);
00455             a+=k[0]+(((uint32_t)k[1])<<16);
00456             break;
00457         case 11:
00458             c+=((uint32_t)k8[10])<<16; /* fall through */
00459             /* no break */
00460         case 10:
00461             c+=k[4];
00462             b+=k[2]+(((uint32_t)k[3])<<16);
00463             a+=k[0]+(((uint32_t)k[1])<<16);
00464             break;
00465         case 9 :
00466             c+=k8[8]; /* fall through */
00467             /* no break */
00468         case 8 :
00469             b+=k[2]+(((uint32_t)k[3])<<16);
00470             a+=k[0]+(((uint32_t)k[1])<<16);
00471             break;
00472         case 7 :
00473             b+=((uint32_t)k8[6])<<16; /* fall through */
00474             /* 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
00581     union {
00582         const void *ptr;
00583         size_t i;
00584     } u; /* needed for Mac Powerbook G4 */
00585
00586     /* Set up the internal state */
00587     a = b = c = 0xdeadbeef + ((uint32_t)length) + *pc;
00588     c += *pb;
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
00592         chunks */
00593         #ifdef VALGRIND
00594             const uint8_t *k8;
00595         #endif
00596         /*----- all but last block: aligned reads and affect 32 bits of
00597         (a,b,c) */
00598         while (length > 12) {
00599             a += k[0];
00600             b += k[1];
00601             c += k[2];
00602             mix(a,b,c);
00603             length -= 12;
00604             k += 3;
00605         }
00606         /*----- handle the last (probably partial)
00607         block */
00608         /*
00609         * "k[2]&0xffffffff" actually reads beyond the end of the string, but
00610         * then masks off the part it's not allowed to read. Because the
00611         * string is aligned, the masked-off tail is in the same word as the
00612         * rest of the string. Every machine with memory protection I've seen
00613         * does it on word boundaries, so is OK with this. But VALGRIND will
00614         * still catch it and complain. The masking trick does make the hash
00615         * noticeably faster for short strings (like English words).
00616         */
00617         #ifndef VALGRIND
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]&0xffffffff;
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]&0xffffffff;
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]&0xffffffff;
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
00719 chunks */
00720             const uint8_t *k8;
00721
00722             /*----- all but last block: aligned reads and different
00723 mixing */
00724             while (length > 12) {
00725                 a += k[0] + ((uint32_t)k[1]<<16);
00726                 b += k[2] + ((uint32_t)k[3]<<16);
00727                 c += k[4] + ((uint32_t)k[5]<<16);
00728                 mix(a,b,c);
00729                 length -= 12;
00730                 k += 6;
00731             }

```



```

00731      /*----- handle the last (probably partial)
00732      block */
00733      k8 = (const uint8_t *)k;
00734      switch(length) {
00735          case 12:
00736              c+=k[4]+(((uint32_t)k[5])<<16);
00737              b+=k[2]+(((uint32_t)k[3])<<16);
00738              a+=k[0]+(((uint32_t)k[1])<<16);
00739              break;
00740          case 11:
00741              c+=((uint32_t)k8[10])<<16;      /* fall through */
00742              /* no break */
00743          case 10:
00744              c+=k[4];
00745              b+=k[2]+(((uint32_t)k[3])<<16);
00746              a+=k[0]+(((uint32_t)k[1])<<16);
00747              break;
00748          case 9 :
00749              c+=k8[8];                      /* fall through */
00750              /* no break */
00751          case 8 :
00752              b+=k[2]+(((uint32_t)k[3])<<16);
00753              a+=k[0]+(((uint32_t)k[1])<<16);
00754              break;
00755          case 7 :
00756              b+=((uint32_t)k8[6])<<16;      /* fall through */
00757              /* no break */
00758          case 6 :
00759              b+=k[2];
00760              a+=k[0]+(((uint32_t)k[1])<<16);
00761              break;
00762          case 5 :
00763              b+=k8[4];                      /* fall through */
00764              /* no break */
00765          case 4 :
00766              a+=k[0]+(((uint32_t)k[1])<<16);
00767              break;
00768          case 3 :
00769              a+=((uint32_t)k8[2])<<16;      /* fall through */
00770              /* no break */
00771          case 2 :
00772              a+=k[0];
00773              break;
00774          case 1 :
00775              a+=k8[0];
00776              break;
00777          case 0 :
00778              *pc=c;
00779              *pb=b;
00780              return; /* zero length strings require no mixing */
00781      }
00782      } else {                                /* need to read the key one byte at a
00783      time */
00784          const uint8_t *k = (const uint8_t *)key;
00785          /*----- all but the last block: affect some 32 bits of
00786      (a,b,c) */
00787          while (length > 12) {
00788              a += k[0];
00789              a += ((uint32_t)k[1])<<8;
00790              a += ((uint32_t)k[2])<<16;
00791              a += ((uint32_t)k[3])<<24;
00792              b += k[4];
00793              b += ((uint32_t)k[5])<<8;
00794              b += ((uint32_t)k[6])<<16;
00795              b += ((uint32_t)k[7])<<24;
00796              c += k[8];
00797              c += ((uint32_t)k[9])<<8;
00798              c += ((uint32_t)k[10])<<16;
00799              c += ((uint32_t)k[11])<<24;
00800              mix(a,b,c);
00801              length -= 12;
00802              k += 12;
00803          }
00804          /*----- last block: affect all 32 bits
00805      of (c) */
00806          switch(length) {                    /* all the case statements fall
00807      through */
00808              case 12:
00809                  c+=((uint32_t)k[11])<<24;
00810                  /* no break */
00811              case 11:
00812                  c+=((uint32_t)k[10])<<16;
00813                  /* no break */
00814              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          * noticeably faster for short strings (like English words).
00897          */
00898 #ifndef VALGRIND
00899         switch(length) {
00900             case 12:
00901                 c+=k[2];

```

```

00903         b+=k[1];
00904         a+=k[0];
00905         break;
00906     case 11:
00907         c+=k[2]&0xffffffff00;
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]&0xffffffff00;
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]&0xffffffff00;
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 #else /* make valgrind happy */
00953     k8 = (const uint8_t *)k;
00954     switch(length) {          /* all the case statements fall
                                through */
00955     case 12:
00956         c+=k[2];
00957         b+=k[1];
00958         a+=k[0];
00959         break;
00960     case 11:
00961         c+=((uint32_t)k8[10])<<8; /* fall through */
00962     case 10:
00963         c+=((uint32_t)k8[9])<<16; /* fall through */
00964     case 9 :
00965         c+=((uint32_t)k8[8])<<24; /* fall through */
00966     case 8 :
00967         b+=k[1];
00968         a+=k[0];
00969         break;
00970     case 7 :
00971         b+=((uint32_t)k8[6])<<8; /* fall through */
00972     case 6 :
00973         b+=((uint32_t)k8[5])<<16; /* fall through */
00974     case 5 :
00975         b+=((uint32_t)k8[4])<<24; /* fall through */
00976     case 4 :
00977         a+=k[0];
00978         break;
00979     case 3 :
00980         a+=((uint32_t)k8[2])<<8; /* fall through */
00981     case 2 :
00982         a+=((uint32_t)k8[1])<<16; /* fall through */
00983     case 1 :
00984         a+=((uint32_t)k8[0])<<24;
00985         break;
00986     }
00987 
```

```

00988         case 0 :
00989             return c;
00990     }
00991 #endif /* !VALGRIND */
00992
00993     } else {
00994         /* need to read the key one byte at a time
00995         */
00996         const uint8_t *k = (const uint8_t *)key;
00997         /*----- all but the last block: affect some 32 bits of
00998         (a,b,c) */
00999         while (length > 12) {
01000             a += ((uint32_t)k[0])<<24;
01001             a += ((uint32_t)k[1])<<16;
01002             a += ((uint32_t)k[2])<<8;
01003             a += ((uint32_t)k[3]);
01004             b += ((uint32_t)k[4])<<24;
01005             b += ((uint32_t)k[5])<<16;
01006             b += ((uint32_t)k[6])<<8;
01007             b += ((uint32_t)k[7]);
01008             c += ((uint32_t)k[8])<<24;
01009             c += ((uint32_t)k[9])<<16;
01010             c += ((uint32_t)k[10])<<8;
01011             c += ((uint32_t)k[11]);
01012             mix(a,b,c);
01013             length -= 12;
01014             k += 12;
01015         }
01016         /*----- last block: affect all 32 bits of
01017         (c) */
01018         switch(length) {
01019             /* all the case statements fall
01020             through */
01021             case 12:
01022                 c+=k[11];
01023                 /* no break */
01024             case 11:
01025                 c+=((uint32_t)k[10])<<8;
01026                 /* no break */
01027             case 10:
01028                 c+=((uint32_t)k[9])<<16;
01029                 /* no break */
01030             case 9 :
01031                 c+=((uint32_t)k[8])<<24;
01032                 /* no break */
01033             case 8 :
01034                 b+=k[7];
01035                 /* no break */
01036             case 7 :
01037                 b+=((uint32_t)k[6])<<8;
01038                 /* no break */
01039             case 6 :
01040                 b+=((uint32_t)k[5])<<16;
01041                 /* no break */
01042             case 5 :
01043                 b+=((uint32_t)k[4])<<24;
01044                 /* no break */
01045             case 4 :
01046                 a+=k[3];
01047                 /* no break */
01048             case 3 :
01049                 a+=((uint32_t)k[2])<<8;
01050                 /* no break */
01051             case 2 :
01052                 a+=((uint32_t)k[1])<<16;
01053                 /* no break */
01054             case 1 :
01055                 a+=((uint32_t)k[0])<<24;
01056                 break;
01057             case 0 :
01058                 return (c);
01059         }
01060     }
01061     final(a,b,c);
01062     return (c);
01063 }
01064 #ifdef SELF_TEST
01065 /* used for timings */
01066 static void driver1() {
01067     uint8_t buf[256];
01068     uint32_t i;
01069     uint32_t h=0;
01070     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
01104             for (j=0; j<8; ++j) { /*----- for each input
01105 bit, */
01106                 for (m=1; m<8; ++m) { /*----- for serveral possible
01107 initvals, */
01108                     for (l=0; l<HASHSTATE; ++l) {
01109                         e[l]=f[l]=g[l]=h[l]=x[l]=y[l]=~((uint32_t)0);
01110
01111                         /*---- check that every output bit is affected by that
01112 input bit */
01113                         for (k=0; k<MAXPAIR; k+=2) {
01114                             uint32_t finished=1;
01115                             /* keys have one bit different */
01116                             for (l=0; l<hlen+1; ++l) {
01117                                 a[l] = b[l] = (uint8_t)0;
01118
01119                                 /* have a and b be two keys differing in only one bit
01120
01121                                 a[i] ^= (k<<j);
01122                                 a[i] ^= (k>>(8-j));
01123                                 c[0] = hashlittle(a, hlen, m);
01124                                 b[i] ^= ((k+1)<<j);
01125                                 b[i] ^= ((k+1)>>(8-j));
01126                                 d[0] = hashlittle(b, hlen, m);
01127                                 /* check every bit is 1, 0, set, and not set at least
01128 once */
01129                                 for (l=0; l<HASHSTATE; ++l) {
01130                                     e[l] &= (c[l]^d[l]);
01131                                     f[l] &= ~(c[l]^d[l]);
01132                                     g[l] &= c[l];
01133                                     h[l] &= ~c[l];
01134                                     x[l] &= d[l];
01135                                     y[l] &= ~d[l];
01136                                     if (e[l]|f[l]|g[l]|h[l]|x[l]|y[l]) {
01137                                         finished=0;
01138                                     }
01139                                 }
01140                                 if (finished) {
01141                                     break;
01142                                 }
01143                             }
01144                             if (k>z) {
01145                                 z=k;
01146                             }
01147                             if (k==MAXPAIR) {
01148                                 printf("Some bit didn't change: ");
01149                                 printf("%.8x %.8x %.8x %.8x %.8x %.8x ",
01150                                     e[0],f[0],g[0],h[0],x[0],y[0]);
01151                                 printf("i %d j %d m %d len %d\n", i, j, m, hlen);
01152                             }
01153                             if (z==MAXPAIR) {
01154                                 goto done;
01155                             }
01156                         }
01157                     }
01158                 }
01159             }
01160         }
01161     }
01162 }

```

```

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 qq[] = "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 qq[] = "xxThis is the time for all good men to come to the aid of
their country...";
01172     uint32_t j;
01173     uint8_t qq[] = "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\n", hashword((const uint32_t *)q, (sizeof(q)-1)/4, 13),
hashword((const uint32_t *)q, (sizeof(q)-5)/4, 13),
hashword((const uint32_t *)q, (sizeof(q)-9)/4, 13));
01179     p = q;
01180     printf("%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x\n",
hashlittle(p, sizeof(q)-1, 13), hashlittle(p,
sizeof(q)-2, 13),
hashlittle(p, sizeof(q)-3, 13), hashlittle(p,
sizeof(q)-4, 13),
hashlittle(p, sizeof(q)-5, 13), hashlittle(p,
sizeof(q)-6, 13),
hashlittle(p, sizeof(q)-7, 13), hashlittle(p,
sizeof(q)-8, 13),
hashlittle(p, sizeof(q)-9, 13), hashlittle(p,
sizeof(q)-10, 13),
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\n",
hashlittle(p, sizeof(q)-1, 13), hashlittle(p,
sizeof(q)-2, 13),
hashlittle(p, sizeof(q)-3, 13), hashlittle(p,
sizeof(q)-4, 13),
hashlittle(p, sizeof(q)-5, 13), hashlittle(p,
sizeof(q)-6, 13),
hashlittle(p, sizeof(q)-7, 13), hashlittle(p,
sizeof(q)-8, 13),
hashlittle(p, sizeof(q)-9, 13), hashlittle(p,
sizeof(q)-10, 13),
hashlittle(p, sizeof(q)-11, 13), hashlittle(p,
sizeof(q)-12, 13));
01198     p = &qq[2];
01199     printf("%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x\n",
hashlittle(p, sizeof(q)-1, 13), hashlittle(p,
sizeof(q)-2, 13),
hashlittle(p, sizeof(q)-3, 13), hashlittle(p,
sizeof(q)-4, 13),
hashlittle(p, sizeof(q)-5, 13), hashlittle(p,
sizeof(q)-6, 13),
hashlittle(p, sizeof(q)-7, 13), hashlittle(p,
sizeof(q)-8, 13),
hashlittle(p, sizeof(q)-9, 13), hashlittle(p,
sizeof(q)-10, 13),
hashlittle(p, sizeof(q)-11, 13), hashlittle(p,
sizeof(q)-12, 13));
01206     p = &qq[3];
01207     printf("%.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x %.8x\n",
hashlittle(p, sizeof(q)-1, 13), hashlittle(p,
sizeof(q)-2, 13),
hashlittle(p, sizeof(q)-3, 13), hashlittle(p,
sizeof(q)-4, 13),
hashlittle(p, sizeof(q)-5, 13), hashlittle(p,
sizeof(q)-6, 13),
hashlittle(p, sizeof(q)-7, 13), hashlittle(p,
sizeof(q)-8, 13),
hashlittle(p, sizeof(q)-9, 13), hashlittle(p,
sizeof(q)-10, 13),
hashlittle(p, sizeof(q)-11, 13), hashlittle(p,
sizeof(q)-12, 13));
01212

```

```

    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     buf[0] = ~0;
01260     for (i=0; i<HASHSTATE; ++i) {
01261         state[i] = 1;
01262     }
01263     printf("These should all be different\n");
01264     for (i=0, h=0; i<8; ++i) {
01265         h = hashlittle(buf, 0, h);
01266         printf("%21d 0-byte strings, hash is  %.8x\n", i, h);
01267     }
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 static int main() {
01292     driver1(); /* test that the key is hashed: used for timings */
01293     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\n"
00084           "    This is free software, and you are welcome to redistribute it\n\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 )) {
00119         if ((ci->flock = lockpidfile(ci->runfile)) < 0)
00120         {
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) && (!
00144 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 #ifndef __WIN32__
00158 /*
00159 * set up signal handler
00160 */
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 interrupts*/
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     /*print 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     /* interrupt 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** { **NFCTTRACK\_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     NFCTTRACK_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,
00061 void *data) {
00062     char buf[1024];
00063
00064     nfct_snprintf(buf, sizeof(buf), ct, NFCT_T_UNKNOWN, NFCT_O_DEFAULT,
00065 NFCT_OF_SHOW_LAYER3 | NFCT_OF_TIME | NFCT_OF_TIMESTAMP);
00066     printf("%s\n", buf);
00067
00068     return (NFCT_CB_CONTINUE);
00069 }
00070
00071 static struct nfct_struct *nf_ctrack_alloc(uint8_t subsys_id,
00072 unsigned subscriptions) {
00073     struct nfct_struct *nfct;
00074
00075     if (!(nfct = objalloc((sizeof *ctrack), close_nfct))) {
00076         return (NULL);
00077     }
00078
00079     /* expectations and conntrack */
00080     if (!(nfct->nfct = nfct_open(subsys_id, subscriptions))) {
00081         objunref(nfct);
00082         return (NULL);
00083     }
00084
00085     if ((nfct->fd = nfct_fd(nfct->nfct)) < 0) {
00086         objunref(nfct);
00087         return (NULL);
00088     }
00089
00090     return (nfct);
00091 }

```

```

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     return (ct);
00130 }
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,
00186 port);
00187             }
00188             break;
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, NFCTTRACK_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 interrupt 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 interrupt 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)) &&
(len = recv(nfq->fd, buf, sizeof(buf), 0)) >= 0) {
00135             objlock(nfq);
00136             nfq_handle_packet(nfq->h, buf, len);
00137             objunlock(nfq);
00138         }
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     {
00178         objunref(nfq);
00179         return (NULL);
00180     }
00181
00182 objlock(nfqueues);
00183 if ((nfqueues->queues || (nfqueues->queues = create_bucketlist
00184 (0, nfqueue_hash))) &&
00185     !addtobucket(nfqueues->queues, nfq)) {
00186     objunref(nfqueues);
00187     objunref(nfq);
00188     return (NULL);
00189 }
00190 objunlock(nfqueues);
00191
00192 framework_mkthread(nfqueue_thread, NULL, NULL, nfq, 0);
00193
00194 return (nfq);
00195 }
00196
00197 static int nfqueue_callback(struct nfq_q_handle *qh, struct nfgenmsg *msg,
00198 struct nfq_data *nfad, void *data) {
00199     struct nfq_queue *nfq_q = data;
00200     unsigned char *pkt;
00201     struct nfqnl_msg_packet_hdr *ph;
00202     void *mangle = NULL;
00203     uint32_t ret, mark;
00204     uint32_t id = 0;
00205     uint32_t len = 0;
00206     uint32_t verdict = NF_DROP;
00207
00208     if ((ph = nfq_get_msg_packet_hdr(nfad)) {
00209         id = ntohs(ph->packet_id);
00210     }
00211     mark = nfq_get_nfmark(nfad);
00212
00213     if ((len = nfq_get_payload(nfad, &pkt)) <= 0) {
00214         pkt = NULL;
00215     }
00216
00217     if (nfq_q->cb) {
00218         verdict = nfq_q->cb(nfad, ph, (char *)pkt, len, nfq_q->data, &
00219 mark, &mangle);
00220     }
00221
00222     if (mangle && !(len = objsize(mangle))) {
00223         objunref(mangle);
00224         mangle = NULL;
00225     }
00226
00227     ret = nfq_set_verdict2(qh, id, verdict, mark, len, (mangle) ? mangle : pkt)
00228 ;
00229
00230     if (mangle) {
00231         objunref(mangle);
00232     }
00233
00234     return (ret);
00235 }
00236
00237 extern struct nfq_queue *nfqueue_attach(uint16_t pf,
00238 uint16_t num, uint8_t mode, uint32_t range, nfqueue_cb cb, void *
00239 data) {
00240     struct nfq_queue *nfq_q;
00241
00242     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 authentication.*
- 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\\_results](#) \* [\\_dtsldapsearch](#) (struct [ldap\\_conn](#) \*ldap, const char \*base, int scope, const char \*filter, char \*\*attrs, int b64enc, int \*err)
- 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 requiring a distinguished 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_rematr` (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](#).

### 14.38.2 Function Documentation

**14.38.2.1** `struct ldap_results * _dtsldapsearch ( struct ldap_conn * ldap, const char * base, int scope, const char * filter, char ** attrs, int b64enc, int * err )` [read]

Definition at line 1559 of file [openldap.c](#).

References [addtobucket\(\)](#), [ldap\\_results::count](#), [create\\_bucketlist\(\)](#), [ldap\\_results::entries](#), [ldap\\_results::first\\_entry](#), [ldap\\_conn::ldap](#), [ldap\\_conn::limit](#), [ldap\\_entry::next](#), [objalloc\(\)](#), [objlock\(\)](#), [objref\(\)](#), [objunlock\(\)](#), [objunref\(\)](#), [ldap\\_entry::prev](#), [ldap\\_conn::sctrlsp](#), and [ldap\\_conn::timelim](#).

Referenced by [ldap\\_search\\_base\(\)](#), [ldap\\_search\\_one\(\)](#), and [ldap\\_search\\_sub\(\)](#).

```

{
    struct timeval timeout;
    struct ldap_results *results;
    struct ldap_entry *lent, *prev = NULL;
    LDAPMessage *result, *message = NULL;
    int res = LDAP_SUCCESS;

    if (!objref(ldap)) {
        if (err) {
            *err = LDAP_UNAVAILABLE;
        }
        if (attrs) {
            free(attrs);
        }
        return NULL;
    }

    if ((results = objalloc(sizeof(*results), free_result))) {
        results->entries = create_bucketlist(4,
            searchresults_hash);
    }

    timeout.tv_sec = ldap->timelim;
    timeout.tv_usec = 0;

    objlock(ldap);
    if (!results || !results->entries ||
        (res = ldap_search_ext_s(ldap->ldap, base, scope, filter, attrs
            , 0, ldap->sctrlsp, NULL, &timeout, ldap->limit, &result))) {
        objunlock(ldap);
        objunref(ldap);
        objunref(results);
        ldap_msgfree(result);
        if (err) {
            *err = (!results || !results->entries) ? LDAP_NO_MEMORY :
res;
        }
        if (attrs) {
            free(attrs);
        }
        return NULL;
    }
    objunlock(ldap);

    if (attrs) {
        free(attrs);
    }

    if ((results->count = ldap_count(ldap->ldap, result, err)) < 0) {
        objunref(ldap);
        objunref(results);
        ldap_msgfree(result);
        return NULL;
    }

    while((lent = ldap_getent(ldap->ldap, &message, result, b64enc,
        err))) {
        if (!results->first_entry) {
            results->first_entry = lent;

```

```

    }
    if (!addtobucket(results->entries, lent)) {
        res = LDAP_NO_MEMORY;
        objunref(lent);
        break;
    }
    lent->next = NULL;
    if (prev) {
        prev->next = lent;
        lent->prev = prev;
    } else {
        lent->prev = NULL;
    }
    prev = lent;
    objunref(lent);
}
ldap_msgfree(result);

if (err) {
    *err = res;
}

if (res) {
    objunref(results);
    results = NULL;
}

objunref(ldap);
return results;
}

```

## 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 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     for(cnt=0; cnt < 3; cnt++) {
00149         if (lmod->bl[cnt]) {
00150             objunref(lmod->bl[cnt]);
00151         }
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 (sas1->mech) {
00172         free((void *)sas1->mech);
00173     }
00174     if (sas1->realm) {
00175         free((void *)sas1->realm);
00176     }
00177     if (sas1->authcid) {
00178         free((void *)sas1->authcid);
00179     }
00180     if (sas1->passwd) {
00181         free((void *)sas1->passwd);
00182     }
00183     if (sas1->authzid) {
00184         free((void *)sas1->authzid);
00185     }
00186 }
00187
00188 static void free_ldapconn(void *data) {
00189     struct ldap_conn *ld = data;
00190
00191     if (ld->uri) {
00192         free(ld->uri);
00193     }
00194     if (ld->ldap) {
00195         ldap_unbind_ext_s(ld->ldap, ld->sctrlrsp, NULL);
00196     }
00197     if (ld->sasl) {
00198         objunref(ld->sasl);
00199     }
00200     if (ld->simple) {
00201         objunref(ld->simple);
00202     }
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     if (ent->dn) {
00224         ldap_memfree((void *)ent->dn);
00225     }
00226     if (ent->rdn) {
00227         objunref(ent->rdn);
00228     }
00229     if (ent->dnufn) {
00230         free((void *)ent->dnufn);
00231     }
00232     if (ent->attrs) {
00233         objunref(ent->attrs);
00234     }
00235     if (ent->first_attr) {
00236         for (la = ent->first_attr; la; la = la->next) {
00237             objunref(la);
00238         }
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 = jenkins(hashkey, strlen(hashkey), 0);
00297     } else {
00298         ret = jenkins(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,
00304     ber_int_t msgid, void *params) {
00305     struct ldap_conn *ldap = params;
00306     int res = LDAP_UNAVAILABLE;
00307
00308     if (!objref(ldap)) {
00309         return LDAP_UNAVAILABLE;
00310     }
00311
00312     if (ldap->sasl) {
00313         int sasl_flags = LDAP_SASL_AUTOMATIC | LDAP_SASL_QUIET;
00314         struct sasl_defaults *sasl = ldap->sasl;
00315
00316         res = ldap_sasl_interactive_bind_s(ld, NULL, sasl->mech, ldap->
00317             sctrlsp, NULL, sasl_flags, dts_sasl_interact, sasl);
00318     } else {
00319         if (ldap->simple) {
00320             struct ldap_simple *simple = ldap->simple;
00321
00322             res = ldap_sasl_bind_s(ld, simple->dn, LDAP_SASL_SIMPLE, simple->
00323                 cred, ldap->sctrlsp, NULL, NULL);
00324         }
00325     }
00326
00327     objunref(ldap);
00328     return res;
00329 }
00330
00331 extern struct ldap_conn *ldap_connect(const char *uri,
00332     enum ldap_starttls starttls, int timelimit, int limit, int
00333     debug, int *err) {
00334     struct ldap_conn *ld;
00335     int version = 3;
00336     int res, sslres;
00337     struct timeval timeout;
00338
00339     if (!(ld = objalloc(sizeof(*ld), free_ldapconn))) {
00340         return NULL;
00341     }
00342
00343     ld->uri = strdup(uri);
00344     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
00551     if (authcid) {
00552         ALLOC_CONST(sasl->authcid, authcid);
00553     } else {
00554         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHCID, &sasl->authcid
00555     );
00556     }
00557
00558     if (authzid) {
00559         ALLOC_CONST(sasl->authzid, authzid);
00560     } else {
00561         ldap_get_option(ld->ldap, LDAP_OPT_X_SASL_AUTHZID, &sasl->authzid
00562     );
00563     }
00564
00565     objlock(ld);
00566     if (ld->sasl) {
00567         objunref(ld->sasl);
00568     }
00569     ld->sasl = sasl;
00570     res = ldap_sasl_interactive_bind_s(ld->ldap, NULL, sasl->mech, ld->
00571     sctrlsp, NULL, sasl_flags, dts_sasl_interact, sasl);
00572     objunlock(ld);
00573     objunref(ld);
00574     return res;
00575 }
00576
00577 extern const char *ldap_errmsg(int res) {
00578     return ldap_err2string(res);
00579 }
00580
00581 static int32_t searchresults_hash(const void *data, int key) {
00582     int ret;
00583     const struct ldap_entry *ent = data;
00584     const char *hashkey = (key) ? data : ent->dn;
00585
00586     if (hashkey) {
00587         ret = jenkins_hash(hashkey, strlen(hashkey), 0);
00588     } else {
00589         ret = jenkins_hash(ent, sizeof(ent), 0);
00590     }
00591     return(ret);
00592 }
00593
00594 extern struct ldap_results *ldap_search_sub(struct
00595 ldap_conn *ld, const char *base, const char *filter, int b64enc,
00596 int *res, ...) {
00597     va_list a_list;
00598     char *attr, **tmp, **attrs = NULL;
00599     int cnt = 1;
00600
00601     va_start(a_list, res);
00602     while ((attr=va_arg(a_list, void *))) {
00603         cnt++;
00604     }
00605     va_end(a_list);
00606
00607     if (cnt > 1) {
00608         tmp = attrs = malloc(sizeof(void *)*cnt);
00609
00610         va_start(a_list, res);
00611         while ((attr=va_arg(a_list, char *))) {
00612             *tmp = attr;
00613             tmp++;
00614         }
00615         va_end(a_list);
00616         *tmp=NULL;
00617     }
00618
00619     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 = jenkins(hashkey, strlen(hashkey), 0);
00824     } else {
00825         ret = jenkins(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, **laval;
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         laval = objalloc(sizeof(void *) * (cnt+1), free_attrvalarr);
00863         if (!laval || !la) {
00864             if (res) {
00865                 *res = LDAP_NO_MEMORY;
00866             }
00867             if (la) {
00868                 objunref(la);
00869             }
00870             if (laval) {
00871                 objunref(laval);
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 = laval;
00881         la->name = attr;
00882         la->count = cnt;
00883
00884         for(; *tmp; tmp++) {
00885             struct berval *bval = *tmp;
00886
00887             *laval = lav = objalloc(sizeof(*lav), free_attrval);
00888             laval++;
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     *laval = 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, dcnt=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                 dcnt++;
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-dcnt)*2+dcnt, 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     return ent;
01053 }
01054
01055 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_rematrr(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_repatrr(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,
01484     const char *attr) {
01485     struct bucket_list *bl = ladd->bl;
01486     struct ldap_modreq *modr = NULL;
01487
01488     if (bl && !(modr = bucket_list_find_key(bl, attr))) {
01489         if (!(modr = new_modreq(bl, attr))) {
01490             return NULL;
01491         }
01492     }
01493     return modr;
01494 }
01500 extern int ldap_add_attr(struct ldap_add *ladd, const char
01501     *attr, ...) {
01502     va_list a_list;
01503     char *val;
01504     struct ldap_modreq *modr;
01505
01506     if (!(modr = getaddreq(ladd, attr))) {
01507         return 1;
01508     }
01509
01510     va_start(a_list, attr);
01511     while((val = va_arg(a_list, void *))) {
01512         if (add_modifyval(modr, val)) {
01513             objunref(modr);
01514             return(1);
01515         }
01516     }
01517
01518     objunref(modr);
01519     va_end(a_list);
01520     return 0;
01521 }
01526 extern int ldap_doaddd(struct ldap_conn *ld, struct ldap_add
01527     *ladd) {
01528     struct bucket_loop *bloop;
01529     struct ldap_modreq *modr;
01530     LDAPMod **modarr, **tmp, *item;
01531     int tot=0, res;
01532
01533     tot = bucket_list_cnt(ladd->bl);
01534     tmp = modarr = calloc(sizeof(void *), (tot+1));
01535
01536     bloop = init_bucket_loop(ladd->bl);
01537     while(bloop && ((modr = next_bucket_loop(bloop))) {
01538         if (!(item = ldap_reqtoarr(modr, -1))) {
01539             ldap_mods_free(modarr, 1);
01540             return LDAP_NO_MEMORY;
01541         }
01542         *tmp = item;
01543         tmp++;
01544         objunref(modr);
01545     }

```



```

01545     objunref(bloop);
01546     *tmp = NULL;
01547
01548     objlock(ld);
01549     res = ldap_modify_ext_s(ld->ldap, ladd->dn, modarr, ld->sctrlrsp
, 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->sctrlrsp, 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@distrotech.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,
    radius_cb read_cb, void *cb_data) {
00342     int scnt;
00343     unsigned char *vector;
00344     unsigned short len;
00345     struct radius_server *server;
00346     struct radius_session *session;
00347     struct radius_connection *connex;
00348     struct bucket_loop *sloop, *cloop;
00349     struct timeval curtime;
00350
00351
00352     gettimeofday(&curtime, NULL);
00353     sloop = init_bucket_loop(servers);
00354     objref(hint);
00355     while (sloop && (server = next_bucket_loop(sloop))) {
00356         objlock(server);
00357         if ((hint && (server->id <= hint->minserver)) ||
            (server->service.tv_sec > curtime.tv_sec)) {
00358             objunlock(server);
00359             objunref(server);
00360             continue;
00361         }
00362         if (!server->connex) {
00363             connex = radconnect(server);
00364             objunref(connex);
00365             objunlock(server);
00366             objref(server);
00367         } else {
00368             objunlock(server);
00369         }
00370         cloop = init_bucket_loop(server->connex);
00371         while (cloop && (connex = next_bucket_loop(cloop))) {
00372             objlock(connex);
00373             if (connex->sessions && (bucket_list_cnt(
    connex->sessions) > 254)) {
00374                 objunlock(connex);
00375                 objunref(connex);
00376                 /* if im overflowing get next or add new*/
00377                 objlock(server);
00378                 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             if (session->passwd) {
00480                 addradattrpasswd(session->packet, session->passwd,
00481 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             objunref(session);
00500         }
00501         objunref(bloop);
00502     }
00503 }
00504
00505 static void radius_rcv(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 );
00547     md5sum2(rtok2, packet, plen, connex->server->secret,
00548     strlen(connex->server->secret));
00549     if (md5cmp(rtok, rtok2)) {
00550         printf("Invalid Signature");
00551         return;
00552     }
00553     if (session->read_cb) {
00554         packet->len = plen;
00555         session->read_cb(packet, session->cb_data);
00556     }
00557     remove_bucket_item(connex->sessions, session);
00558     objunref(session);
00559 }
00560
00561 static void *rad_return(void *data) {
00562     struct radius_connection *connex = data;
00563     fd_set rd_set, act_set;
00564     struct timeval tv;
00565     int selfd;
00566
00567     FD_ZERO(&rd_set);
00568     FD_SET(connex->socket->sock, &rd_set);
00569
00570     while (framework_threadok()) {
00571         act_set = rd_set;
00572         tv.tv_sec = 0;
00573         tv.tv_usec = 200000;
00574
00575         selfd = select(connex->socket->sock + 1, &act_set, NULL, NULL
00576 , &tv);
00577         if ((selfd < 0 && errno == EINTR) || (!selfd)) {
00578             rad_resend(connex);
00579             continue;
00580         } else {
00581             if (selfd < 0) {
00582                 break;
00583             }
00584
00585             if (FD_ISSET(connex->socket->sock, &act_set)) {
00586                 radius_rcv(data);
00587             }
00588             rad_resend(connex);
00589         }
00590     }
00591     return NULL;
00592 }
00593
00594 static void del_radconnect(void *data) {
00595     struct radius_connection *connex = data;
00596
00597     objunref(connex->server);
00598     objunref(connex->sessions);
00599     objunref(connex->socket);
00600 }
00601
00602 static struct radius_connection *radconnect(struct
00603 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,
00609 server->authport, NULL))) {
00610             if (!server->connex) {
00611                 server->connex = create_bucketlist(0,
00612 hash_connex);
00613             }
00614             setsockopt(connex->socket->sock, SOL_IP, IP_RECVERR, (char
00615 *)&val, sizeof(val));
00616             connex->server = server;
00617             genrand(&connex->id, sizeof(connex->id));
00618             addtobucket(server->connex, connex);
00619             framework_mkthread(rad_return, NULL, NULL, connex
00620 , 0);
00621         }
00622     }
00623     return connex;
00624 }
00625
00626 extern unsigned char *radius_attr_first(struct radius_packet
00627 *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](#) 0xdead0de  
*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 0xdead0de

```

```

00035
00036 /* ref counted objects*/
00037 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         {
00169             ref->cnt++;
00170             ret = ref->cnt;
00171         }
00172         pthread_mutex_unlock(&ref->lock);
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     {
00400         lhead = lhead->next;
00401     }
00402     return (lhead);
00403 }
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 = jenkinshash(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->gethash(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@distrotech.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, esum, 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 [dtls\\_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 to default 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
00019 #include <stdint.h>
00020 #ifdef __WIN32__
00021 #include <winsock2.h>
00022 #include <windows.h>
00023 #include <ws2tcpip.h>
00024 #endif
00025 #include <openssl/ssl.h>
00026 #include <openssl/err.h>
00027 #include <sys/stat.h>
00028 #include <unistd.h>
00029 #ifndef __WIN32__
00030 #include <sys/socket.h>
00031 #include <arpa/inet.h>
00032 #endif
00033 #include "include/dtsapp.h"
00034
00035 enum SSLFLAGS {
00036     SSL_TLSV1 = 1 << 0,
00037     SSL_SSLV2 = 1 << 1,
00038     SSL_SSLV3 = 1 << 2,
00039     SSL_DTLSV1 = 1 << 3,
00040     SSL_CLIENT = 1 << 4,
00041     SSL_SERVER = 1 << 5,
00042     SSL_DTLSCON = 1 << 6
00043 };
00044
00045 struct ssldata {
00046     SSL_CTX *ctx;
00047     SSL *ssl;
00048     BIO *bio;
00049     int flags;
00050     const SSL_METHOD *meth;
00051     struct ssldata *parent;
00052 };
00053
00054 #define COOKIE_SECRET_LENGTH 32
00055 static unsigned char *cookie_secret = NULL;
00056
00057 static int generate_cookie(SSL *ssl, unsigned char *cookie, unsigned int *
00058 cookie_len) {
00059     union sockstruct peer;
00060     if (!ssl || !cookie_secret || (*cookie_len < COOKIE_SECRET_LENGTH
00061 )) {
00062         return (0);
00063     }
00064 }

```

```

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     objlock(ssl);
00187
00188     while (ssl->ssl && (ret = _ssl_shutdown(ssl) && (cnt < 3))) {
00189         selfd = socket_select(sock, ret);
00190         if (selfd <= 0) {
00191             break;
00192         }
00193         cnt++;
00194     }
00195     if (ssl->ssl) {
00196         SSL_free(ssl->ssl);
00197         ssl->ssl = NULL;
00198     }
00199     objunlock(ssl);
00200 }
00201
00202 static void free_ssldata(void *data) {
00203     struct ssldata *ssl = data;
00204
00205     if (ssl->parent) {
00206         objunref(ssl->parent);
00207     }
00208
00209     if (ssl->ctx) {
00210         SSL_CTX_free(ssl->ctx);
00211         ssl->ctx = NULL;
00212     }
00213 }
00214
00215 static int verify_callback (int ok, X509_STORE_CTX *ctx) {
00216     return (1);
00217 }
00218
00219 static struct ssldata *sslinit(const char *cacert, const char *cert,
00220                                const char *key, int verify, const SSL_METHOD *meth, int flags) {
00221     struct ssldata *ssl;
00222     struct stat finfo;
00223     int ret = -1;
00224
00225     if (!(ssl = objalloc(sizeof(*ssl), free_ssldata))) {
00226         return NULL;
00227     }
00228
00229     ssl->flags = flags;
00230     ssl->meth = meth;
00231     if (!(ssl->ctx = SSL_CTX_new(meth))) {
00232         objunref(ssl);
00233         return NULL;
00234     }
00235
00236     if (!stat(cacert, &finfo)) {
00237         if (S_ISDIR(finfo.st_mode) && (SSL_CTX_load_verify_locations(ssl->ctx,
00238                                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,
00247                                SSL_FILETYPE_PEM) == 1)) {
00248         ret = 0;
00249     }
00250     if (!ret && (SSL_CTX_use_PrivateKey_file(ssl->ctx, key, SSL_FILETYPE_PEM)
00251 == 1)) {
00252         ret = 0;
00253     }
00254     if (!ret && (SSL_CTX_check_private_key (ssl->ctx) == 1)) {
00255         ret = 0;
00256     }
00257
00258     /*XXX Should create a tmp 512 bit rsa key for RSA ciphers also need DH
00259     http://www.openssl.org/docs/ssl/SSL_CTX_set_cipher_list.html
00260     SSL_CTX_set_cipher_list*/
00261
00262     if (!ret) {
00263         /* XXX CRL verification
00264         X509_VERIFY_PARAM *param;
00265         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 *tls1_init(const char *cacert, const char *cert, const
char *key, int verify) {
00288     const SSL_METHOD *meth = TLS1_method();
00289
00290     return (sslini(cacert, cert, key, verify, meth, SSL_TLS1));
00291 }
00292
00293 #ifndef OPENSSL_NO_SSL2
00299 extern void *ssl2_init(const char *cacert, const char *cert, const
char *key, int verify) {
00300     const SSL_METHOD *meth = SSL2_method();
00301
00302     return (sslini(cacert, cert, key, verify, meth, SSL_SSL2));
00303 }
00304 #endif
00305
00311 extern void *ssl3_init(const char *cacert, const char *cert, const
char *key, int verify) {
00312     const SSL_METHOD *meth = SSL3_method();
00313     struct ssl_data *ssl;
00314
00315     ssl = sslini(cacert, cert, key, verify, meth, SSL_SSL3);
00316
00317     return (ssl);
00318 }
00319
00325 extern void *dtls1_init(const char *cacert, const char *cert, const
char *key, int verify) {
00326     const SSL_METHOD *meth = DTLS1_method();
00327     struct ssl_data *ssl;
00328
00329     ssl = sslini(cacert, cert, key, verify, meth, SSL_DTLS1);
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 ssl_data *
orig, int accept) {
00337     struct ssl_data *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 EBADE:
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     if (ssl && ssl->ssl) {
00567         objlock(ssl);
00568         if (SSL_state(ssl->ssl) != SSL_ST_OK) {
00569             objunlock(ssl);
00570             return (SSL_ERROR_SSL);
00571         }
00572         ret = SSL_write(ssl->ssl, buf, num);
00573         err = SSL_get_error(ssl->ssl, ret);
00574         objunlock(ssl);
00575     }

```

```

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 dtls_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@distrotech.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 jenkinshash(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     switch(sig) {
00166     case SIGHUP:
00167         clearflag(thread, TL_THREAD_RUN);
00168         break;
00169     case SIGINT:
00170     case SIGTERM:
00171         clearflag(thread, TL_THREAD_RUN);
00172         setflag(thread, TL_THREAD_STOP);
00173     }
00174     objunref(thread);
00175     return 1;
00176     #else
00177     return 0;
00178     #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
00264 extern int startthreads(void) {
00265     struct threadcontainer *tc;
00266
00267     tc = (objref(threads)) ? threads : NULL;
00268
00269     if (tc) {
00270         objunref(tc);
00271         return 1;
00272     }
00273
00274     if (!(tc = objalloc(sizeof(*threads), close_threads))) {
00275         return 0;
00276     }
00277
00278     if (!(tc->list && !(tc->list = create_bucketlist(4,
hash_thread))) {
00279         objunref(tc);
00280         return 0;
00281     }
00282
00283     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 shutting 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 socket 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     FD_ZERO(&rd_set);
00088
00089     fd = sock->sock;
00090     fdf = fcntl(fd, F_GETFL);
00091     fcntl(fd, F_SETFD, fdf | O_NONBLOCK);
00092     /*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 interrupt 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     sock = unsock->sock;
00176     sock->flags |= SOCK_FLAG_UNIX;
00177     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 interrupt 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     close(fd);
00247     objunref(unsock);
00248
00249     return NULL;
00250 }
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->saddr;
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 beginning 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,
00098     unsigned long len, const void *data2, unsigned long len2) {
00099     SHA512_CTX c;
00100
00101     SHA512_Init(&c);
00102     SHA512_Update(&c, data, len);
00103     if (data2) {
00104         SHA512_Update(&c, data2, len2);
00105     }
00106     SHA512_Final(buff, &c);
00107 }
00114 extern void sha512sum(unsigned char *buff, const void *data, unsigned
00115     long len) {
00116     sha512sum2(buff, data, len, NULL, 0);
00117 }
00118
00127 extern void sha256sum2(unsigned char *buff, const void *data,
00128     unsigned long len, const void *data2, unsigned long len2) {
00129     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 shalsum2(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 shalsum(unsigned char *buff, const void *data, unsigned long
    len) {
00174     shalsum2(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 shalcmp(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], ickey[64];
00260     int bcnt;
00261
00262     memset(ickey, 0, 64);
00263     memset(okey, 0, 64);
00264
00265     if (klen < 64) {
00266         memcpy(ickey, key, klen);

```

```

00267     memcpy(okey, key, klen);
00268 } else {
00269     md5sum(okey, key, klen);
00270     memcpy(ikkey, okey, klen);
00271 }
00272
00273 for (bcnt = 0; bcnt < 64; bcnt++) {
00274     ikkey[bcnt] ^= 0x36;
00275     okey[bcnt] ^= 0x5c;
00276 };
00277
00278 func(buff, ikkey, 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 shal1hmac(unsigned char *buff, const void *data, unsigned
    long len, const void *key, unsigned long klen) {
00303     _hmac(buff, data, len, key, klen, shal1sum2, 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
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
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 = (PIP_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         i++;
00030     } while (i <= tries);
00031

```

```

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 tmp4[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     if (ainfo) {
00149         free(ainfo);
00150     }
00151     return ifinf;
00152 }
00153 }
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);
}

#ifdef __WIN32
void unixsockettest(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) {
#ifdef __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);
    }
#ifdef __WIN32
    } else if (!strcmp(argv[1], "unix_d")) {
        unixsockettest(argv[2], SOCK_DGRAM);
    } else if (!strcmp(argv[1], "unix_s")) {
        unixsockettest(argv[2], SOCK_STREAM);
    }
#endif
    } else {
        printf("Invalid Option\n");
    }
}

```



```
    }  
}
```

# Index

- [\\_dtsldapsearch](#)
    - [openldap.c, 411](#)
- [ALLOC\\_CONST](#)
  - [Distrotech Application Library, 29](#)
- [accept\\_func](#)
  - [doxygen/examples/socket.c, 300](#)
- [accept\\_socket](#)
  - [Network socket interface, 65](#)
- [acctport](#)
  - [radius\\_server, 277](#)
- [add\\_radserver](#)
  - [Radius client interface, 118](#)
- [addr](#)
  - [fwsocket, 255](#)
- [addradattrint](#)
  - [Radius client interface, 118](#)
- [addradattrip](#)
  - [Radius client interface, 118](#)
- [addradattrstr](#)
  - [Radius client interface, 119](#)
- [addtobucket](#)
  - [Hashed bucket linked lists of referenced objects, 47](#)
- [adji](#)
  - [natmap, 271](#)
- [adjo](#)
  - [natmap, 271](#)
- [attr](#)
  - [ldap\\_modreq, 267](#)
- [attrs](#)
  - [ldap\\_entry, 264](#)
  - [radius\\_packet, 276](#)
  - [xml\\_node, 293](#)
- [authcid](#)
  - [sasl\\_defaults, 282](#)
- [authport](#)
  - [radius\\_server, 277](#)
- [authzid](#)
  - [sasl\\_defaults, 282](#)
- [b64enc](#)
  - [Miscellaneous utilities., 123](#)
- [b64enc\\_buf](#)
  - [Miscellaneous utilities., 123](#)
- [basic\\_auth, 243](#)
  - [passwd, 243](#)
  - [user, 243](#)
- [bio](#)
  - [ssldata, 286](#)
- [bl](#)
  - [ldap\\_add, 260](#)
  - [ldap\\_modify, 266](#)
- [blist](#)
  - [bucket\\_loop, 247](#)
- [blist\\_cb](#)
  - [Hashed bucket linked lists of referenced objects, 46](#)
- [blist\\_obj, 244](#)
  - [data, 244](#)
  - [hash, 244](#)
  - [next, 244](#)
  - [prev, 244](#)
- [blisthash](#)
  - [Hashed bucket linked lists of referenced objects, 46](#)
- [body](#)
  - [curlbuf, 251](#)
- [bsize](#)
  - [curlbuf, 251](#)
- [bucket](#)
  - [bucket\\_loop, 247](#)
- [bucket\\_list, 245](#)
  - [bucketbits, 245](#)
  - [count, 245](#)
  - [hash\\_func, 245](#)
  - [list, 246](#)
  - [locks, 246](#)
  - [version, 246](#)
- [bucket\\_list\\_cnt](#)
  - [Hashed bucket linked lists of referenced objects, 48](#)
- [bucket\\_list\\_find\\_key](#)
  - [Hashed bucket linked lists of referenced objects, 49](#)
- [bucket\\_loop, 246](#)
  - [blist, 247](#)
  - [bucket, 247](#)
  - [cur, 247](#)
  - [cur\\_hash, 247](#)
  - [head, 247](#)
  - [head\\_hash, 247](#)
  - [version, 247](#)
- [bucketbits](#)
  - [bucket\\_list, 245](#)
- [bucketlist\\_callback](#)
  - [Hashed bucket linked lists of referenced objects, 49](#)
- [buf](#)
  - [ipaddr\\_req, 258](#)
  - [iplink\\_req, 259](#)
- [buff](#)
  - [zobj, 297](#)
- [buffer](#)
  - [ldap\\_attrval, 262](#)

- Burtle Bob hash algorithim., 209
  - final, 210
  - HASH\_BIG\_ENDIAN, 210
  - HASH\_LITTLE\_ENDIAN, 211
  - hashbig, 212
  - hashlittle, 215
  - hashlittle2, 218
  - hashmask, 211
  - hashsize, 211
  - hashword, 222
  - hashword2, 223
  - JHASH\_INITVAL, 211
  - jenhash, 211
  - mix, 211
  - rot, 212
- c\_type
  - curlbuf, 251
- CURL Url interface., 196
  - curl\_authcb, 197
  - curl\_buf2xml, 198
  - curl\_geturl, 199
  - curl\_newauth, 199
  - curl\_newpost, 200
  - curl\_post, 197
  - curl\_postitem, 200
  - curl\_posturl, 201
  - curl\_progress\_func, 197
  - curl\_progress\_newdata, 198
  - curl\_progress\_pause, 198
  - curl\_setauth\_cb, 201
  - curl\_setprogress, 202
  - curl\_ungzip, 202
  - curlclose, 203
  - curlinit, 203
  - url\_escape, 204
  - url\_unescape, 204
- cat
  - config\_file, 249
- cb
  - nfq\_queue, 272
- cb\_data
  - radius\_session, 279
- check\_ipv4
  - IPv4 functions, 142
- checkipv6mask
  - IPv6 functions, 150
- checksum
  - Micelaneous utilities., 124
- checksum\_add
  - Micelaneous utilities., 124
- children
  - fwsocket, 255
- cidrcnt
  - IPv4 functions, 142
- cidrtosn
  - IPv4 functions, 142
- cleanup
  - socket\_handler, 284
  - thread\_pvt, 288
- clearflag
  - Referenced Lockable Objects, 37
- client
  - socket\_handler, 284
  - unixclient\_sockthread, 290
- client\_func
  - doxygen/examples/socket.c, 300
- close\_socket
  - Network socket interface, 66
- closenetlink
  - Linux network interface functions, 95
- cnt
  - ldap\_modreq, 267
  - ref\_obj, 281
  - xml\_node\_iter, 294
- code
  - radius\_packet, 276
- config\_cat\_callback
  - INI Style config file Interface, 108
- config\_catchb
  - INI Style config file Interface, 108
- config\_category, 248
  - entries, 248
  - name, 248
- config\_entry, 248
  - item, 249
  - value, 249
- config\_entry\_callback
  - INI Style config file Interface, 109
- config\_entrycb
  - INI Style config file Interface, 108
- config\_file, 249
  - cat, 249
  - filename, 250
  - filepath, 250
- config\_file\_callback
  - INI Style config file Interface, 109
- config\_filecb
  - INI Style config file Interface, 108
- connect
  - socket\_handler, 284
- Connection Tracking, 234
  - NFCTrack\_DONE, 235
  - NF\_CTRACK\_FLAGS, 235
  - nf\_ctrack\_buildct, 235
  - nf\_ctrack\_close, 235
  - nf\_ctrack\_delete, 236
  - nf\_ctrack\_dump, 236
  - nf\_ctrack\_endtrace, 236
  - nf\_ctrack\_init, 237
  - nf\_ctrack\_nat, 237
  - nf\_ctrack\_trace, 237
  - nfct\_struct, 234
- connex
  - radius\_server, 278
- count
  - bucket\_list, 245

- ldap\_attr, 260
  - ldap\_results, 269
- create\_bucketlist
  - Hashed bucket linked lists of referenced objects, 50
- create\_kernmac
  - Linux network interface functions, 95
- create\_kernvlan
  - Linux network interface functions, 96
- create\_tun
  - Linux network interface functions, 97
- cred
  - ldap\_simple, 270
- ctx
  - ssldata, 286
- cur
  - bucket\_loop, 247
- cur\_hash
  - bucket\_loop, 247
- curl\_authcb
  - CURL Url interface., 197
- curl\_buf2xml
  - CURL Url interface., 198
- curl\_geturl
  - CURL Url interface., 199
- curl\_newauth
  - CURL Url interface., 199
- curl\_newpost
  - CURL Url interface., 200
- curl\_post, 250
  - CURL Url interface., 197
  - first, 250
  - last, 250
- curl\_postitem
  - CURL Url interface., 200
- curl\_posturl
  - CURL Url interface., 201
- curl\_progress\_func
  - CURL Url interface., 197
- curl\_progress\_newdata
  - CURL Url interface., 198
- curl\_progress\_pause
  - CURL Url interface., 198
- curl\_setauth\_cb
  - CURL Url interface., 201
- curl\_setprogress
  - CURL Url interface., 202
- curl\_unzip
  - CURL Url interface., 202
- curlbuf, 251
  - body, 251
  - bsize, 251
  - c\_type, 251
  - header, 251
  - hsize, 252
- curlclose
  - CURL Url interface., 203
- curlinit
  - CURL Url interface., 203
- curpos
  - xml\_node\_iter, 294
- DTS\_OJBREF\_CLASS
  - Referenced Lockable Objects, 37
- daddr
  - pseudohdr, 274
- daemonize
  - Distrotech Application Library, 30
- data
  - blist\_obj, 244
  - nfq\_queue, 272
  - ref\_obj, 281
  - socket\_handler, 284
  - thread\_pvt, 288
  - unixclient\_sockthread, 290
  - unixserv\_sockthread, 291
- delete\_kernmac
  - Linux network interface functions, 97
- delete\_kernvlan
  - Linux network interface functions, 98
- destroy
  - ref\_obj, 281
- developer
  - framework\_core, 253
- Distrotech Application Library, 27
  - ALLOC\_CONST, 29
  - daemonize, 30
  - 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
  - sys sighandler, 30
- Distrotech Application Library (Todo), 232
- dn
  - ldap\_add, 260
  - ldap\_entry, 264
  - ldap\_modify, 266
  - ldap\_simple, 270
- dnufn
  - ldap\_entry, 264
- doc
  - xslt\_doc, 296
- doxygen/dox/buckets.dox, 299
- doxygen/dox/examples.dox, 299
- doxygen/dox/index.dox, 299
- doxygen/dox/main.dox, 299
- doxygen/dox/modules.dox, 299
- doxygen/dox/refobj.dox, 299
- doxygen/dox/sockets.dox, 299
- doxygen/dox/sockex.dox, 299
- doxygen/dox/thread.dox, 299
- doxygen/examples/socket.c, 299, 304

- accept\_func, [300](#)
- client\_func, [300](#)
- FRAMEWORK\_MAIN, [301](#)
- server\_func, [302](#)
- socktest, [302](#)
- unixsocktest, [303](#)
- dtls\_listenssl
  - SSL socket support, [79](#)
- dtlshandltimeout
  - SSL socket support, [80](#)
- dtlstimeout
  - SSL socket support, [80](#)
- dtlsv1\_init
  - SSL socket support, [81](#)
- dtls\_serveropts
  - SSL socket support, [81](#)
- email
  - framework\_core, [253](#)
- endpoint
  - unixclient\_sockthread, [290](#)
- entries
  - config\_category, [248](#)
  - ldap\_results, [269](#)
- epre
  - natmap, [271](#)
- eui48to64
  - IPv6 functions, [151](#)
- 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, [301](#)
- fd
  - nfq\_struct, [273](#)
- File utility functions, [154](#)
  - is\_dir, [154](#)
  - is\_exec, [154](#)
  - is\_file, [155](#)
  - mk\_dir, [155](#)
- filename
  - config\_file, [250](#)
- filepath
  - config\_file, [250](#)
- final
  - Burtle Bob hash algorithim., [210](#)
- first
  - curl\_post, [250](#)
  - ldap\_modreq, [267](#)
- first\_attr
  - ldap\_entry, [265](#)
- first\_entry
  - ldap\_results, [269](#)
- flags
  - framework\_core, [253](#)
  - fwsocket, [255](#)
  - nfq\_struct, [273](#)
  - ssldata, [286](#)
  - thread\_pvt, [288](#)
- flock
  - framework\_core, [253](#)
- framework\_core, [252](#)
  - developer, [253](#)
  - email, [253](#)
  - flags, [253](#)
  - flock, [253](#)
  - progname, [253](#)
  - runfile, [253](#)
  - sa, [253](#)
  - sig\_handler, [253](#)
  - www, [254](#)
  - year, [254](#)
- 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, [288](#)
- fwsocket, [254](#)
  - addr, [255](#)
  - children, [255](#)
  - flags, [255](#)
  - parent, [255](#)
  - proto, [255](#)
  - sock, [255](#)
  - ssl, [256](#)
  - type, [256](#)
- genrand
  - Micelaneous utilities., [124](#)
- get\_category\_loop
  - INI Style config file Interface, [109](#)
- get\_category\_next
  - INI Style config file Interface, [110](#)
- get\_config\_category
  - INI Style config file Interface, [110](#)
- get\_config\_entry
  - INI Style config file Interface, [111](#)
- get\_config\_file
  - INI Style config file Interface, [111](#)
- get\_iface\_index
  - Linux network interface functions, [98](#)
- get\_ifinfo
  - Windows Support, [229](#)
- get\_ifipaddr

- Linux network interface functions, 99
- get\_ip6\_addrprefix
  - IPv6 functions, 151
- getbcaddr
  - IPv4 functions, 143
- getfirstaddr
  - IPv4 functions, 144
- getlastaddr
  - IPv4 functions, 144
- getnetaddr
  - IPv4 functions, 145
- gzinflatebuf
  - Zlib Interface, 206
- h
  - nfq\_struct, 273
- HASH\_BIG\_ENDIAN
  - Burtle Bob hash algorithim., 210
- HASH\_LITTLE\_ENDIAN
  - Burtle Bob hash algorithim., 211
- hash
  - blist\_obj, 244
- hash\_func
  - bucket\_list, 245
- hashbig
  - Burtle Bob hash algorithim., 212
- Hashed bucket linked lists of referenced objects, 45
  - addtobucket, 47
  - blist\_cb, 46
  - blisthash, 46
  - bucket\_list\_cnt, 48
  - bucket\_list\_find\_key, 49
  - bucketlist\_callback, 49
  - create\_bucketlist, 50
  - init\_bucket\_loop, 51
  - next\_bucket\_loop, 51
  - remove\_bucket\_item, 52
  - remove\_bucket\_loop, 53
- Hashing and digest functions, 129
- hashlittle
  - Burtle Bob hash algorithim., 215
- hashlittle2
  - Burtle Bob hash algorithim., 218
- hashmask
  - Burtle Bob hash algorithim., 211
- hashsize
  - Burtle Bob hash algorithim., 211
- hashword
  - Burtle Bob hash algorithim., 222
- hashword2
  - Burtle Bob hash algorithim., 223
- head
  - bucket\_loop, 247
- head\_hash
  - bucket\_loop, 247
- header
  - curlbuf, 251
- hsize
  - curlbuf, 252

- i
  - ipaddr\_req, 258
  - iplink\_req, 259
- IP\_PROTO\_V4
  - IPv4 and IPv6 functions, 138
- IP\_PROTO\_V6
  - IPv4 and IPv6 functions, 138
- IPV4\_SCORE\_RESERVED
  - Linux network interface functions, 94
- IPV4\_SCORE\_ROUTABLE
  - Linux network interface functions, 94
- IPV4\_SCORE\_ZEROCONF
  - Linux network interface functions, 94
- IPV6\_SCORE\_RESERVED
  - Linux network interface functions, 94
- IPV6\_SCORE\_ROUTABLE
  - Linux network interface functions, 94
- IPV6\_SCORE\_SIXIN4
  - Linux network interface functions, 94
- IPv4 and IPv6 functions
  - IP\_PROTO\_V4, 138
  - IP\_PROTO\_V6, 138
- INI Style config file Interface, 107
  - config\_cat\_callback, 108
  - config\_catcb, 108
  - config\_entry\_callback, 109
  - config\_entrycb, 108
  - config\_file\_callback, 109
  - config\_filecb, 108
  - get\_category\_loop, 109
  - get\_category\_next, 110
  - get\_config\_category, 110
  - get\_config\_entry, 111
  - get\_config\_file, 111
  - process\_config, 112
  - unrefconfigfiles, 113
- IPv4 and IPv6 functions, 138
  - inet\_lookup, 139
  - ipversion, 138
  - packetchecksum, 139
- IPv4 functions, 141
  - check\_ip4, 142
  - cidrcnt, 142
  - cidrtosn, 142
  - getbcaddr, 143
  - getfirstaddr, 144
  - getlastaddr, 144
  - getnetaddr, 145
  - ipv4checksum, 146
  - ipv4icmpchecksum, 146
  - ipv4tcpchecksum, 146
  - ipv4udpchecksum, 147
  - packetchecksumv4, 147
  - reservedip, 148
  - score\_ip4, 149
- IPv6 functions, 150
  - checkipv6mask, 150
  - eui48to64, 151

- get\_ip6\_addrprefix, [151](#)
  - ipv6to4prefix, [152](#)
  - packetchecksumv6, [152](#)
  - score\_ipv6, [153](#)
- IPv6 Nat Mapping, [225](#)
  - natmap, [225](#)
  - rfc6296\_map, [225](#)
  - rfc6296\_map\_add, [226](#)
  - rfc6296\_test, [227](#)
- id
  - radius\_connection, [275](#)
  - radius\_packet, [276](#)
  - radius\_server, [278](#)
  - radius\_session, [279](#)
- idx
  - ifinfo, [257](#)
- ifaddr
  - ifinfo, [257](#)
- ifdown
  - Linux network interface functions, [100](#)
- ifhwaddr
  - Linux network interface functions, [100](#)
- ifinfo, [256](#)
  - idx, [257](#)
  - ifaddr, [257](#)
  - ipv4addr, [257](#)
  - ipv6addr, [257](#)
- ifrename
  - Linux network interface functions, [101](#)
- ifup
  - Linux network interface functions, [101](#)
- inet\_lookup
  - IPv4 and IPv6 functions, [139](#)
- inet\_ntop
  - Windows Support, [230](#)
- init\_bucket\_loop
  - Hashed bucket linked lists of referenced objects, [51](#)
- interface\_bind
  - Linux network interface functions, [102](#)
- ipaddr\_req, [257](#)
  - buf, [258](#)
  - i, [258](#)
  - n, [258](#)
- iplink\_req, [258](#)
  - buf, [259](#)
  - i, [259](#)
  - n, [259](#)
- ipre
  - natmap, [271](#)
- ipv4\_score
  - Linux network interface functions, [94](#)
- ipv4addr
  - ifinfo, [257](#)
- ipv4checksum
  - IPv4 functions, [146](#)
- ipv4icmpchecksum
  - IPv4 functions, [146](#)
- ipv4tcpchecksum
  - IPv4 functions, [146](#)
- ipv4udpchecksum
  - IPv4 functions, [147](#)
- ipv6\_score
  - Linux network interface functions, [94](#)
- ipv6addr
  - ifinfo, [257](#)
- ipv6to4prefix
  - IPv6 functions, [152](#)
- ipversion
  - IPv4 and IPv6 functions, [138](#)
- is\_dir
  - File utility functions, [154](#)
- is\_exec
  - File utility functions, [154](#)
- is\_file
  - File utility functions, [155](#)
- is\_gzip
  - Zlib Interface, [207](#)
- item
  - config\_entry, [249](#)
- JHASH\_INITVAL
  - Burtle Bob hash algorithm., [211](#)
- jenhash
  - Burtle Bob hash algorithm., [211](#)
- jointhreads
  - Posix thread interface, [59](#)
- key
  - xml\_node, [293](#)
- LDAP\_ATTRTYPE\_B64
  - Openldap/SASL Interface, [159](#)
- LDAP\_ATTRTYPE\_CHAR
  - Openldap/SASL Interface, [159](#)
- LDAP\_ATTRTYPE\_OCTET
  - Openldap/SASL Interface, [159](#)
- LDAP\_STARTTLS\_ATTEMPT
  - Openldap/SASL Interface, [159](#)
- LDAP\_STARTTLS\_ENFORCE
  - Openldap/SASL Interface, [160](#)
- LDAP\_STARTTLS\_NONE
  - Openldap/SASL Interface, [159](#)
- last
  - curl\_post, [250](#)
  - ldap\_modreq, [267](#)
- ldap
  - ldap\_conn, [263](#)
- ldap\_add, [259](#)
  - bl, [260](#)
  - dn, [260](#)
  - Openldap/SASL Interface, [159](#)
- ldap\_add\_attr
  - Openldap/SASL Interface, [160](#)
- ldap\_addinit
  - Openldap/SASL Interface, [160](#)
- ldap\_attr, [260](#)
  - count, [260](#)

- name, [260](#)
- next, [261](#)
- prev, [261](#)
- vals, [261](#)
- ldap\_attrtype
  - Openldap/SASL Interface, [159](#)
- ldap\_attrval, [261](#)
  - buffer, [262](#)
  - len, [262](#)
  - type, [262](#)
- ldap\_conn, [262](#)
  - ldap, [263](#)
  - limit, [263](#)
  - Openldap/SASL Interface, [159](#)
  - sasl, [263](#)
  - sctrlsp, [263](#)
  - simple, [263](#)
  - timelim, [263](#)
  - uri, [263](#)
- ldap\_connect
  - Openldap/SASL Interface, [161](#)
- ldap\_doaddd
  - Openldap/SASL Interface, [162](#)
- ldap\_domodify
  - Openldap/SASL Interface, [162](#)
- ldap\_entry, [264](#)
  - attrs, [264](#)
  - dn, [264](#)
  - dnufn, [264](#)
  - first\_attr, [265](#)
  - list, [265](#)
  - next, [265](#)
  - prev, [265](#)
  - rdn, [265](#)
  - rdncnt, [265](#)
- ldap\_errmsg
  - Openldap/SASL Interface, [163](#)
- ldap\_getattr
  - Openldap/SASL Interface, [164](#)
- ldap\_getentry
  - Openldap/SASL Interface, [164](#)
- ldap\_mod\_add
  - Openldap/SASL Interface, [164](#)
- ldap\_mod\_addattr
  - Openldap/SASL Interface, [165](#)
- ldap\_mod\_del
  - Openldap/SASL Interface, [166](#)
- ldap\_mod\_delattr
  - Openldap/SASL Interface, [166](#)
- ldap\_mod\_rematrr
  - Openldap/SASL Interface, [167](#)
- ldap\_mod\_rep
  - Openldap/SASL Interface, [167](#)
- ldap\_mod\_repattr
  - Openldap/SASL Interface, [168](#)
- ldap\_modify, [265](#)
  - bl, [266](#)
  - dn, [266](#)
  - Openldap/SASL Interface, [159](#)
- ldap\_modifyinit
  - Openldap/SASL Interface, [168](#)
- ldap\_modreq, [266](#)
  - attr, [267](#)
  - cnt, [267](#)
  - first, [267](#)
  - last, [267](#)
- ldap\_modval, [267](#)
  - next, [267](#)
  - value, [267](#)
- ldap\_rdn, [268](#)
  - name, [268](#)
  - next, [268](#)
  - prev, [268](#)
  - value, [268](#)
- ldap\_results, [269](#)
  - count, [269](#)
  - entries, [269](#)
  - first\_entry, [269](#)
- ldap\_saslbind
  - Openldap/SASL Interface, [169](#)
- ldap\_search\_base
  - Openldap/SASL Interface, [170](#)
- ldap\_search\_one
  - Openldap/SASL Interface, [171](#)
- ldap\_search\_sub
  - Openldap/SASL Interface, [172](#)
- ldap\_simple, [270](#)
  - cred, [270](#)
  - dn, [270](#)
- ldap\_simplebind
  - Openldap/SASL Interface, [172](#)
- ldap\_simplerebind
  - Openldap/SASL Interface, [173](#)
- ldap\_starttls
  - Openldap/SASL Interface, [159](#)
- ldap\_unref\_attr
  - Openldap/SASL Interface, [174](#)
- ldap\_unref\_entry
  - Openldap/SASL Interface, [174](#)
- len
  - ldap\_attrval, [262](#)
  - pseudohdr, [274](#)
  - radius\_packet, [276](#)
- limit
  - ldap\_conn, [263](#)
- Linux Netfilter, [233](#)
- Linux network interface functions, [93](#)
  - closenethlink, [95](#)
  - create\_kernmac, [95](#)
  - create\_kernvlan, [96](#)
  - create\_tun, [97](#)
  - delete\_kernmac, [97](#)
  - delete\_kernvlan, [98](#)
  - get\_iface\_index, [98](#)
  - get\_ifipaddr, [99](#)
  - IPV4\_SCORE\_RESERVED, [94](#)



- IPV4\_SCORE\_ROUTABLE, [94](#)
- IPV4\_SCORE\_ZEROCONF, [94](#)
- IPV6\_SCORE\_RESERVED, [94](#)
- IPV6\_SCORE\_ROUTABLE, [94](#)
- IPV6\_SCORE\_SIXIN4, [94](#)
- ifdown, [100](#)
- ifhwaddr, [100](#)
- ifrename, [101](#)
- ifup, [101](#)
- interface\_bind, [102](#)
- ipv4\_score, [94](#)
- ipv6\_score, [94](#)
- randhwaddr, [102](#)
- set\_interface\_addr, [103](#)
- set\_interface\_flags, [103](#)
- set\_interface\_ipaddr, [104](#)
- set\_interface\_name, [105](#)
- list
  - bucket\_list, [246](#)
  - ldap\_entry, [265](#)
  - threadcontainer, [289](#)
- lock
  - ref\_obj, [281](#)
- lockpidfile
  - Distrotech Application Library, [33](#)
- locks
  - bucket\_list, [246](#)
- ltrim
  - Micelaneous utilities., [125](#)
- MD5 Hashing and digest functions, [130](#)
  - md5cmp, [130](#)
  - md5hmac, [130](#)
  - md5sum, [131](#)
  - md5sum2, [131](#)
- magic
  - ref\_obj, [281](#)
- make\_socket
  - Network socket interface, [66](#)
- manager
  - threadcontainer, [289](#)
- mask
  - natmap, [271](#)
  - unixserv\_sockthread, [291](#)
- mcast4\_ip
  - Multicast sockets, [89](#)
- mcast6\_ip
  - Multicast sockets, [89](#)
- mcast\_socket
  - Multicast sockets, [90](#)
- md5cmp
  - MD5 Hashing and digest functions, [130](#)
- md5hmac
  - MD5 Hashing and digest functions, [130](#)
- md5sum
  - MD5 Hashing and digest functions, [131](#)
- md5sum2
  - MD5 Hashing and digest functions, [131](#)
- mech
  - sasl\_defaults, [283](#)
- meth
  - ssldata, [286](#)
- Micelaneous utilities., [122](#)
  - b64enc, [123](#)
  - b64enc\_buf, [123](#)
  - checksum, [124](#)
  - checksum\_add, [124](#)
  - genrand, [124](#)
  - ltrim, [125](#)
  - rtrim, [125](#)
  - seedrand, [126](#)
  - strlenzero, [126](#)
  - touch, [126](#)
  - trim, [127](#)
  - tvtontp64, [127](#)
  - verifysum, [128](#)
- minserver
  - radius\_session, [279](#)
- mix
  - Burtle Bob hash algorithim., [211](#)
- mk\_dir
  - File utility functions, [155](#)
- Multicast sockets, [89](#)
  - mcast4\_ip, [89](#)
  - mcast6\_ip, [89](#)
  - mcast\_socket, [90](#)
- n
  - ipaddr\_req, [258](#)
  - iplink\_req, [259](#)
- NFCTrack\_DONE
  - Connection Tracking, [235](#)
- NFQUEUE\_DONE
  - Queue interface, [240](#)
- NF\_CTRACK\_FLAGS
  - Connection Tracking, [235](#)
- NF\_QUEUE\_FLAGS
  - Queue interface, [240](#)
- name
  - config\_category, [248](#)
  - ldap\_attr, [260](#)
  - ldap\_rdn, [268](#)
  - radius\_server, [278](#)
  - xml\_attr, [292](#)
  - xml\_node, [293](#)
  - xslt\_param, [297](#)
- natmap, [270](#)
  - adji, [271](#)
  - adjo, [271](#)
  - epre, [271](#)
  - ipre, [271](#)
  - IPv6 Nat Mapping, [225](#)
  - mask, [271](#)
- Network socket interface, [63](#)
  - accept\_socket, [65](#)
  - close\_socket, [66](#)
  - make\_socket, [66](#)
  - 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, 67
- sockconnect, 68
- socketclient, 68
- socketread, 69
- socketread\_d, 69
- socketrecv, 64
- socketserver, 71
- socketwrite, 71
- socketwrite\_d, 72
- tcpbind, 74
- tcpconnect, 74
- udpbind, 75
- udpconnect, 75
- new\_radpacket
  - Radius client interface, 119
- next
  - blist\_obj, 244
  - ldap\_attr, 261
  - ldap\_entry, 265
  - ldap\_modval, 267
  - ldap\_rdn, 268
- next\_bucket\_loop
  - Hashed bucket linked lists of referenced objects, 51
- nf\_track\_buildct
  - Connection Tracking, 235
- nf\_track\_close
  - Connection Tracking, 235
- nf\_track\_delete
  - Connection Tracking, 236
- nf\_track\_dump
  - Connection Tracking, 236
- nf\_track\_endtrace
  - Connection Tracking, 236
- nf\_track\_init
  - Connection Tracking, 237
- nf\_track\_nat
  - Connection Tracking, 237
- nf\_track\_trace
  - Connection Tracking, 237
- nfct\_struct
  - Connection Tracking, 234
- nfq
  - nfq\_queue, 272
- nfq\_data
  - Queue interface, 239
- nfq\_queue, 272
  - cb, 272
  - data, 272
  - nfq, 272
  - num, 272
  - qh, 272
  - Queue interface, 239
- nfq\_struct, 272
  - fd, 273
  - flags, 273
  - h, 273
  - pf, 273
- nfqnl\_msg\_packet\_hdr
  - Queue interface, 240
- nfqueue\_attach
  - Queue interface, 240
- nfqueue\_cb
  - Queue interface, 240
- nodeptr
  - xml\_node, 293
- nodes
  - xml\_search, 295
- num
  - nfq\_queue, 272
- objalloc
  - Referenced Lockable Objects, 38
- objchar
  - Referenced Lockable Objects, 39
- objcnt
  - Referenced Lockable Objects, 40
- objdestroy
  - Referenced Lockable Objects, 38
- objlock
  - Referenced Lockable Objects, 40
- objref
  - Referenced Lockable Objects, 41
- objsize
  - Referenced Lockable Objects, 41
- objtrylock
  - Referenced Lockable Objects, 42
- objunlock
  - Referenced Lockable Objects, 42
- objunref
  - Referenced Lockable Objects, 43
- olen
  - radius\_session, 279
  - zobj, 297
- openldap.c
  - \_dtsldapsearch, 411
- Openldap/SASL Interface
  - LDAP\_ATTRTYPE\_B64, 159
  - LDAP\_ATTRTYPE\_CHAR, 159
  - LDAP\_ATTRTYPE\_OCTET, 159
  - LDAP\_STARTTLS\_ATTEMPT, 159
  - LDAP\_STARTTLS\_ENFORCE, 160
  - LDAP\_STARTTLS\_NONE, 159
- Openldap/SASL Interface, 157
  - ldap\_add, 159
  - ldap\_add\_attr, 160
  - ldap\_addinit, 160
  - ldap\_attrtype, 159
  - ldap\_conn, 159
  - ldap\_connect, 161
  - ldap\_doadd, 162
  - ldap\_domodify, 162

- ldap\_errmsg, 163
- ldap\_getattr, 164
- ldap\_getentry, 164
- ldap\_mod\_add, 164
- ldap\_mod\_addattr, 165
- ldap\_mod\_del, 166
- ldap\_mod\_delattr, 166
- ldap\_mod\_remattrib, 167
- ldap\_mod\_rep, 167
- ldap\_mod\_repattrib, 168
- ldap\_modify, 159
- ldap\_modifyinit, 168
- ldap\_saslbind, 169
- ldap\_search\_base, 170
- ldap\_search\_one, 171
- ldap\_search\_sub, 172
- ldap\_simplebind, 172
- ldap\_simplerebind, 173
- ldap\_starttls, 159
- ldap\_unref\_attr, 174
- ldap\_unref\_entry, 174
- packet
  - radius\_session, 280
- packetchecksum
  - IPv4 and IPv6 functions, 139
- packetchecksumv4
  - IPv4 functions, 147
- packetchecksumv6
  - IPv6 functions, 152
- params
  - xslt\_doc, 296
- parent
  - fwsocket, 255
  - ssldata, 287
- passwd
  - basic\_auth, 243
  - radius\_session, 280
  - sasl\_defaults, 283
- pf
  - nfq\_struct, 273
- 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, 244
  - ldap\_attr, 261
  - ldap\_entry, 265
  - ldap\_rdn, 268
- printgnu
  - Distrotech Application Library, 34
- process\_config
  - INI Style config file Interface, 112
- progname
  - framework\_core, 253
- proto
  - fwsocket, 255
  - pseudohdr, 274
- protocol
  - unixserv\_sockthread, 291
- pseudohdr, 273
  - daddr, 274
  - len, 274
  - proto, 274
  - saddr, 274
  - zero, 274
- qh
  - nfq\_queue, 272
- Queue interface, 239
  - NFQUEUE\_DONE, 240
  - NF\_QUEUE\_FLAGS, 240
  - nfq\_data, 239
  - nfq\_queue, 239
  - nfqnl\_msg\_packet\_hdr, 240
  - nfqueue\_attach, 240
  - nfqueue\_cb, 240
  - snprintf\_pkt, 241
- RAD\_CODE\_ACCTREQUEST
  - Radius client interface, 117
- RAD\_CODE\_ACCTRESPONSE
  - Radius client interface, 117
- RAD\_CODE\_AUTHACCEPT
  - Radius client interface, 117
- RAD\_CODE\_AUTHCHALLENGE
  - Radius client interface, 117
- RAD\_CODE\_AUTHREJECT
  - Radius client interface, 117
- RAD\_CODE\_AUTHREQUEST
  - Radius client interface, 117
- RAD\_ATTR\_ACCTID
  - Radius client interface, 115
- RAD\_ATTR\_EAP
  - Radius client interface, 115
- RAD\_ATTR\_MESSAGE

- Radius client interface, 115
- RAD\_ATTR\_NAS\_PORT
  - Radius client interface, 116
- RAD\_ATTR\_PORT\_TYPE
  - Radius client interface, 116
- RAD\_ATTR\_USER\_NAME
  - Radius client interface, 116
- RAD\_AUTH\_HDR\_LEN
  - Radius client interface, 116
- RAD\_AUTH\_TOKEN\_LEN
  - Radius client interface, 116
- RAD\_MAX\_PASS\_LEN
  - Radius client interface, 117
- RADIUS\_CODE
  - Radius client interface, 117
- REFOBJ\_MAGIC
  - Referenced Lockable Objects, 37
- Radius client interface, 114
  - add\_radserver, 118
  - addradattrint, 118
  - addradattrip, 118
  - addradattrstr, 119
  - new\_radpacket, 119
  - RAD\_CODE\_ACCTREQUEST, 117
  - RAD\_CODE\_ACCTRESPONSE, 117
  - RAD\_CODE\_AUTHACCEPT, 117
  - RAD\_CODE\_AUTHCHALLENGE, 117
  - RAD\_CODE\_AUTHREJECT, 117
  - RAD\_CODE\_AUTHREQUEST, 117
  - RAD\_ATTR\_ACCTID, 115
  - RAD\_ATTR\_EAP, 115
  - RAD\_ATTR\_MESSAGE, 115
  - RAD\_ATTR\_NAS\_PORT, 116
  - RAD\_ATTR\_PORT\_TYPE, 116
  - RAD\_ATTR\_USER\_NAME, 116
  - RAD\_AUTH\_HDR\_LEN, 116
  - RAD\_AUTH\_TOKEN\_LEN, 116
  - RAD\_MAX\_PASS\_LEN, 117
  - RADIUS\_CODE, 117
  - radius\_attr\_first, 120
  - radius\_attr\_next, 120
  - radius\_cb, 117
  - radius\_packet, 117
  - send\_radpacket, 120
- radius\_attr\_first
  - Radius client interface, 120
- radius\_attr\_next
  - Radius client interface, 120
- radius\_cb
  - Radius client interface, 117
- radius\_connection, 274
  - id, 275
  - server, 275
  - sessions, 275
  - socket, 275
- radius\_packet, 275
  - attrs, 276
  - code, 276
  - id, 276
  - len, 276
  - Radius client interface, 117
  - token, 276
- radius\_server, 277
  - acctport, 277
  - authport, 277
  - connex, 278
  - id, 278
  - name, 278
  - secret, 278
  - service, 278
  - timeout, 278
- radius\_session, 279
  - cb\_data, 279
  - id, 279
  - minserver, 279
  - olen, 279
  - packet, 280
  - passwd, 280
  - read\_cb, 280
  - request, 280
  - retries, 280
  - sent, 280
- randhwaddr
  - Linux network interface functions, 102
- rdn
  - ldap\_entry, 265
- rdncnt
  - ldap\_entry, 265
- read
  - unixserv\_sockthread, 291
- read\_cb
  - radius\_session, 280
- realm
  - sasl\_defaults, 283
- ref\_obj, 280
  - cnt, 281
  - data, 281
  - destroy, 281
  - lock, 281
  - magic, 281
  - size, 282
- Referenced Lockable Objects, 36
  - clearflag, 37
  - DTS\_OJBREF\_CLASS, 37
  - objalloc, 38
  - objchar, 39
  - objcnt, 40
  - objdestroy, 38
  - objlock, 40
  - objref, 41
  - objsize, 41
  - objtrylock, 42
  - objunlock, 42
  - objunref, 43
  - REFOBJ\_MAGIC, 37
  - refobj\_offset, 38

- setflag, [38](#)
  - testflag, [38](#)
- refobj\_offset
  - Referenced Lockable Objects, [38](#)
- remove\_bucket\_item
  - Hashed bucket linked lists of referenced objects, [52](#)
- remove\_bucket\_loop
  - Hashed bucket linked lists of referenced objects, [53](#)
- request
  - radius\_session, [280](#)
- reservedip
  - IPv4 functions, [148](#)
- retries
  - radius\_session, [280](#)
- rfc6296\_map
  - IPv6 Nat Mapping, [225](#)
- rfc6296\_map\_add
  - IPv6 Nat Mapping, [226](#)
- rfc6296\_test
  - IPv6 Nat Mapping, [227](#)
- rot
  - Burtle Bob hash algorithim., [212](#)
- rtrim
  - Micelaneous utilities., [125](#)
- runfile
  - framework\_core, [253](#)
- 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\_DTL SV1, [78](#)
  - SSL\_SERVER, [79](#)
  - SSL\_SSLV2, [78](#)
  - SSL\_SSLV3, [78](#)
  - SSL\_TL SV1, [78](#)
- SSL\_CLIENT
  - SSL socket support, [78](#)
- SSL\_DTLSCON
  - SSL socket support, [79](#)
- SSL\_DTL SV1
  - SSL socket support, [78](#)
- SSL\_SERVER
  - SSL socket support, [79](#)
- SSL\_SSLV2
  - SSL socket support, [78](#)
- SSL\_SSLV3
  - SSL socket support, [78](#)
- SSL\_TL SV1
  - SSL socket support, [78](#)
- SHA1 Hashing and digest functions, [132](#)
  - sha1cmp, [132](#)
  - sha1hmac, [132](#)
  - sha1sum, [133](#)
  - sha1sum2, [133](#)
- SHA2-256 Hashing and digest functions, [134](#)
  - sha256cmp, [134](#)
  - sha256hmac, [134](#)
  - sha256sum, [135](#)
  - sha256sum2, [135](#)
- SHA2-512 Hashing and digest functions, [136](#)
  - sha512cmp, [136](#)
  - sha512hmac, [136](#)
  - sha512sum, [137](#)
  - sha512sum2, [137](#)
- SSL socket support, [77](#)
  - dtls\_listenssl, [79](#)
  - dtlshandltimeout, [80](#)
  - dtlstimeout, [80](#)
  - dtlsv1\_init, [81](#)
  - dtls\_serveropts, [81](#)
  - SSLFLAGS, [78](#)
  - ssl\_shutdown, [82](#)
  - ssldata, [78](#)
  - sslstartup, [82](#)
  - sslv2\_init, [83](#)
  - sslv3\_init, [83](#)
  - startsslclient, [83](#)
  - tlaccept, [84](#)
  - tlsv1\_init, [84](#)
- SSLFLAGS
  - SSL socket support, [78](#)
- sa
  - framework\_core, [253](#)
  - sockstruct, [285](#)
- sa4
  - sockstruct, [285](#)
- sa6
  - sockstruct, [285](#)
- saddr
  - pseudohdr, [274](#)
- sasl
  - ldap\_conn, [263](#)
- sasl\_defaults, [282](#)
  - authcid, [282](#)
  - authzid, [282](#)
  - mech, [283](#)
  - passwd, [283](#)
  - realm, [283](#)
- score\_ipv4
  - IPv4 functions, [149](#)
- score\_ipv6
  - IPv6 functions, [153](#)
- sctrlsp
  - ldap\_conn, [263](#)
- secret
  - radius\_server, [278](#)
- seedrand

- Miscellaneous utilities., 126
- send\_radpacket
  - Radius client interface, 120
- sent
  - radius\_session, 280
- server
  - radius\_connection, 275
- server\_func
  - doxygen/examples/socket.c, 302
- service
  - radius\_server, 278
- sessions
  - radius\_connection, 275
- set\_interface\_addr
  - Linux network interface functions, 103
- set\_interface\_flags
  - Linux network interface functions, 103
- set\_interface\_ipaddr
  - Linux network interface functions, 104
- set\_interface\_name
  - Linux network interface functions, 105
- setflag
  - Referenced Lockable Objects, 38
- sha1cmp
  - SHA1 Hashing and digest functions, 132
- sha1hmac
  - SHA1 Hashing and digest functions, 132
- sha1sum
  - SHA1 Hashing and digest functions, 133
- sha1sum2
  - SHA1 Hashing and digest functions, 133
- sha256cmp
  - SHA2-256Hashing and digest functions, 134
- sha256hmac
  - SHA2-256Hashing and digest functions, 134
- sha256sum
  - SHA2-256Hashing and digest functions, 135
- sha256sum2
  - SHA2-256Hashing and digest functions, 135
- sha512cmp
  - SHA2-512 Hashing and digest functions, 136
- sha512hmac
  - SHA2-512 Hashing and digest functions, 136
- sha512sum
  - SHA2-512 Hashing and digest functions, 137
- sha512sum2
  - SHA2-512 Hashing and digest functions, 137
- sig\_handler
  - framework\_core, 253
- sighandler
  - thread\_pvt, 288
- simple
  - ldap\_conn, 263
- size
  - ref\_obj, 282
- snprintf\_pkt
  - Queue interface, 241
- sock
  - fwsocket, 255
  - socket\_handler, 284
  - unixclient\_sockthread, 290
  - unixserv\_sockthread, 291
- sock\_flags
  - Network socket interface, 65
- sockaddr2ip
  - Network socket interface, 67
- sockbind
  - Network socket interface, 67
- sockconnect
  - Network socket interface, 68
- socket
  - radius\_connection, 275
- socket\_handler, 283
  - cleanup, 284
  - client, 284
  - connect, 284
  - data, 284
  - sock, 284
- socketclient
  - Network socket interface, 68
- socketread
  - Network socket interface, 69
- socketread\_d
  - Network socket interface, 69
- socketrecv
  - Network socket interface, 64
- socketserver
  - Network socket interface, 71
- socketwrite
  - Network socket interface, 71
- socketwrite\_d
  - Network socket interface, 72
- sockpath
  - unixserv\_sockthread, 292
- sockstruct, 284
  - sa, 285
  - sa4, 285
  - sa6, 285
  - ss, 285
  - un, 285
- sockettest
  - doxygen/examples/socket.c, 302
- src/config.c, 314, 315
- src/curl.c, 319, 321
- src/fileutil.c, 326, 327
- src/include/dtsapp.h, 328, 341
- src/interface.c, 350, 352
- src/iputil.c, 361, 362
- src/libxml2.c, 368, 369
- src/libxslt.c, 377, 378
- src/lookup3.c, 381, 382
- src/main.c, 396
- src/nf\_ctrack.c, 400, 401
- src/nf\_queue.c, 404, 405
- src/openldap.c, 409, 412
- src/radius.c, 430, 431

- src/refobj.c, [438](#), [439](#)
- src/rfc6296.c, [447](#)
- src/socket.c, [306](#), [307](#)
- src/sslutil.c, [449](#), [451](#)
- src/thread.c, [460](#), [461](#)
- src/unixsock.c, [466](#), [467](#)
- src/util.c, [471](#), [473](#)
- src/winiface.cpp, [477](#)
- src/zlib.c, [479](#), [480](#)
- ss
  - sockstruct, [285](#)
- ssl
  - fwsocket, [256](#)
  - ssldata, [287](#)
- ssl\_shutdown
  - SSL socket support, [82](#)
- ssldata, [286](#)
  - bio, [286](#)
  - ctx, [286](#)
  - flags, [286](#)
  - meth, [286](#)
  - parent, [287](#)
  - ssl, [287](#)
  - SSL socket support, [78](#)
- sslstartup
  - SSL socket support, [82](#)
- sslv2\_init
  - SSL socket support, [83](#)
- sslv3\_init
  - SSL socket support, [83](#)
- startsslclient
  - SSL socket support, [83](#)
- startthreads
  - Posix thread interface, [60](#)
- stopthreads
  - Posix thread interface, [60](#)
- strlenzero
  - Miscellaneous utilities., [126](#)
- sys sighandler
  - 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, [74](#)
- tcpconnect
  - Network socket interface, [74](#)
- testflag
  - Referenced Lockable Objects, [38](#)
- thr
  - thread\_pvt, [288](#)
- thread\_can\_start
  - Posix thread interface, [62](#)
- thread\_option\_flags
  - Posix thread interface, [57](#)
- thread\_pvt, [287](#)
  - cleanup, [288](#)
  - data, [288](#)
  - flags, [288](#)
  - func, [288](#)
  - sighandler, [288](#)
  - thr, [288](#)
- thread\_signal
  - Posix thread interface, [61](#)
- threadcleanup
  - Posix thread interface, [56](#)
- threadcontainer, [289](#)
  - list, [289](#)
  - manager, [289](#)
- threadfunc
  - Posix thread interface, [56](#)
- threadopt
  - Posix thread interface, [57](#)
- threads
  - Posix thread interface, [62](#)
- threadsighandler
  - Posix thread interface, [56](#)
- timelim
  - ldap\_conn, [263](#)
- timeout
  - radius\_server, [278](#)
- tlsaccept
  - SSL socket support, [84](#)
- tlsv1\_init
  - SSL socket support, [84](#)
- token
  - radius\_packet, [276](#)
- touch
  - Miscellaneous utilities., [126](#)
- trim
  - Miscellaneous utilities., [127](#)
- tvtontp64
  - Miscellaneous utilities., [127](#)
- type
  - fwsocket, [256](#)
  - ldap\_attrval, [262](#)
- udpbinding

- Network socket interface, [75](#)
- udpconnect
  - Network socket interface, [75](#)
- un
  - sockstruct, [285](#)
- Unix domain sockets, [86](#)
  - unixsocket\_client, [86](#)
  - unixsocket\_server, [87](#)
- unixclient\_sockthread, [289](#)
  - client, [290](#)
  - data, [290](#)
  - endpoint, [290](#)
  - sock, [290](#)
- unixserv\_sockthread, [290](#)
  - data, [291](#)
  - mask, [291](#)
  - protocol, [291](#)
  - read, [291](#)
  - sock, [291](#)
  - sockpath, [292](#)
- unixsocket\_client
  - Unix domain sockets, [86](#)
- unixsocket\_server
  - Unix domain sockets, [87](#)
- unixsocktest
  - doxygen/examples/socket.c, [303](#)
- unrefconfigfiles
  - INI Style config file Interface, [113](#)
- uri
  - ldap\_conn, [263](#)
- url\_escape
  - CURL Url interface., [204](#)
- url\_unescape
  - CURL Url interface., [204](#)
- user
  - basic\_auth, [243](#)
- vals
  - ldap\_attr, [261](#)
- value
  - config\_entry, [249](#)
  - ldap\_modval, [267](#)
  - ldap\_rdn, [268](#)
  - xml\_attr, [292](#)
  - xml\_node, [293](#)
  - xslt\_param, [297](#)
- verifysum
  - Micelaneous utilities., [128](#)
- version
  - bucket\_list, [246](#)
  - bucket\_loop, [247](#)
- Windows Support, [229](#)
  - get\_ifinfo, [229](#)
  - inet\_ntop, [230](#)
- www
  - framework\_core, [254](#)
- XML Interface, [176](#)
  - xml\_addnode, [178](#)
  - xml\_appendnode, [179](#)
  - xml\_close, [179](#)
  - xml\_createpath, [179](#)
  - xml\_delete, [181](#)
  - xml\_doc, [177](#)
  - xml\_doctobuffer, [181](#)
  - xml\_free\_buffer, [181](#)
  - xml\_getattr, [182](#)
  - xml\_getbuffer, [182](#)
  - xml\_getfirstnode, [183](#)
  - xml\_getnextnode, [183](#)
  - xml\_getnode, [184](#)
  - xml\_getnodes, [184](#)
  - xml\_getrootname, [185](#)
  - xml\_getrootnode, [185](#)
  - xml\_init, [186](#)
  - xml\_loadbuf, [186](#)
  - xml\_loaddoc, [186](#)
  - xml\_modify, [187](#)
  - xml\_node, [177](#)
  - xml\_nodecount, [187](#)
  - xml\_savefile, [188](#)
  - xml\_search, [178](#)
  - xml\_setattr, [188](#)
  - xml\_unlink, [189](#)
  - xml\_xpath, [189](#)
- XSLT Interface, [191](#)
  - xslt\_addparam, [192](#)
  - xslt\_apply, [192](#)
  - xslt\_apply\_buffer, [193](#)
  - xslt\_clearparam, [193](#)
  - xslt\_close, [194](#)
  - xslt\_doc, [191](#)
  - xslt\_init, [194](#)
  - xslt\_open, [194](#)
- xml\_addnode
  - XML Interface, [178](#)
- xml\_appendnode
  - XML Interface, [179](#)
- xml\_attr, [292](#)
  - name, [292](#)
  - value, [292](#)
- xml\_close
  - XML Interface, [179](#)
- xml\_createpath
  - XML Interface, [179](#)
- xml\_delete
  - XML Interface, [181](#)
- xml\_doc
  - XML Interface, [177](#)
- xml\_doctobuffer
  - XML Interface, [181](#)
- xml\_free\_buffer
  - XML Interface, [181](#)
- xml\_getattr
  - XML Interface, [182](#)
- xml\_getbuffer



- XML Interface, [182](#)
- xml\_getfirstnode
  - XML Interface, [183](#)
- xml\_getnextnode
  - XML Interface, [183](#)
- xml\_getnode
  - XML Interface, [184](#)
- xml\_getnodes
  - XML Interface, [184](#)
- xml\_getrootname
  - XML Interface, [185](#)
- xml\_getrootnode
  - XML Interface, [185](#)
- xml\_init
  - XML Interface, [186](#)
- xml\_loadbuf
  - XML Interface, [186](#)
- xml\_loaddoc
  - XML Interface, [186](#)
- xml\_modify
  - XML Interface, [187](#)
- xml\_node, [293](#)
  - attrs, [293](#)
  - key, [293](#)
  - name, [293](#)
  - nodeptr, [293](#)
  - value, [293](#)
  - XML Interface, [177](#)
- xml\_node\_iter, [294](#)
  - cnt, [294](#)
  - curpos, [294](#)
  - xsearch, [294](#)
- xml\_nodecount
  - XML Interface, [187](#)
- xml\_savefile
  - XML Interface, [188](#)
- xml\_search, [295](#)
  - nodes, [295](#)
  - XML Interface, [178](#)
  - xmldoc, [295](#)
  - xpathObj, [295](#)
- xml\_setattr
  - XML Interface, [188](#)
- xml\_unlink
  - XML Interface, [189](#)
- xml\_xpath
  - XML Interface, [189](#)
- xmldoc
  - xml\_search, [295](#)
- xpathObj
  - xml\_search, [295](#)
- xsearch
  - xml\_node\_iter, [294](#)
- xslt\_addparam
  - XSLT Interface, [192](#)
- xslt\_apply
  - XSLT Interface, [192](#)
- xslt\_apply\_buffer
  - XSLT Interface, [193](#)
- xslt\_clearparam
  - XSLT Interface, [193](#)
- xslt\_close
  - XSLT Interface, [194](#)
- xslt\_doc, [296](#)
  - doc, [296](#)
  - params, [296](#)
  - XSLT Interface, [191](#)
- xslt\_init
  - XSLT Interface, [194](#)
- xslt\_open
  - XSLT Interface, [194](#)
- xslt\_param, [296](#)
  - name, [297](#)
  - value, [297](#)
- year
  - framework\_core, [254](#)
- zcompress
  - Zlib Interface, [207](#)
- zero
  - pseudohdr, [274](#)
- zlen
  - zobj, [298](#)
- Zlib Interface, [206](#)
  - gzinflatebuf, [206](#)
  - is\_gzip, [207](#)
  - zcompress, [207](#)
  - zuncompress, [208](#)
- zobj, [297](#)
  - buff, [297](#)
  - olen, [297](#)
  - zlen, [298](#)
- zuncompress
  - Zlib Interface, [208](#)