Data Tables

APPENDIX

A

IN THIS APPENDIX

This appendix identifies and lists all the tables and data formats relevant to sockets programming.

Domains: First Parameter to `socket()`

Table A.1 lists the values for the first parameter of the `socket()` system call. You can use these types as well in a `bind()` system call. While most programs use the AF style for both `socket()` and `bind()`, the correct form is to use the PF style for `socket()` and the AF style for `bind()`. If you are uncomfortable using the PF style, you can safely use the AF style because the C header files define the AF style as the PF style. The structure definitions are located in `<bits/socket.h>`.

TABLE A.1 Protocol Family Values for the Domain Parameter of `socket()`

Type	*Description and Example*	*Affiliated Data*

`PF_UNSPEC`	Unspecified	`struct sockaddr {`
		`  unsigned short int sa_family;`
		`  unsigned char  sa_data[14];`
		`};`

`PF_LOCAL`
`PF_UNIX`
`PF_FILE`
	BSD method for accessing local named pipes
	
	`#include <linux/un.h>`
	`struct sockaddr_un addr;`
	`addr.sun_family = AF_UNIX;`
	`strcpy(addr.sun_path,`
	`  "/tmp/mysocket");`
		`#define UNIX_PATH_MAX  108`
		
		`struct sockaddr_un {`
		`  sa_family_t sun_family;`
		`  char sun_path[UNIX_PATH_MAX];`
		`};`

`PF_INET`	Internet IPv4 protocol family
	
	`#include <linux/in.h>`
	`struct sockaddr_in addr;`
	`bzero(&addr, sizeof(addr));`
	`addr.sin_family = AF_INET;`
	`addr.sin_port = htons(9999);`
	`if ( inet_aton("127.0.0.1",`
	`  &addr.sin_addr) == 0 )`
	`  perror("Addr conversion");`
		`struct sockaddr_in {`
		`  sa_family_t  sin family;`
		`  unsigned short int sin_port;`
		`  struct in_addr sin_addr;`
		`  unsigned char  pad[];`
		`};`

TABLE A.1 Continued

Type	*Description and Example*	*Affiliated Data*

AF_AX25	Amateur Radio AX.25
	
	#include <linux/AX25.h>
		typedef struct {
		  char ax25_call[7];
		} ax25_address;
		
		struct sockaddr_ax25 {
		  sa_family_t sax25_family;
		  ax25_address sax25_call;
		  int sax25_ndigis;
		};

PF_IPX	Novell Internet Protocol
	
	#include <linux/ipx.h>
		struct sockaddr_ipx
		{
		  sa_family_t sipx_family;
		  __u16 sipx_port;
		  __u32 sipx_network;
		  unsigned char
		  sipx_node[IPX_NODE_LEN];
		  __u8 sipx_type;
		  /* padding */
		  unsigned char sipx_zero;
		};

PF_APPLETALK	Appletalk DDP
	
	#include <linux/atalk.h>
		struct sockaddr_at {
		  sa_family_t sat_family;
		  u8 sat_port;
		  struct at_addr {
		  u16 s_net;
		  u8 s_node;
		  } sat_addr;
		  char sat_zero[];
		};

TABLE A.1 Continued

Type	*Description and Example*	*Affiliated Data*
PF_NETROM	Amateur Radio NetROM	
PF_BRIDGE	Multiprotocol Bridge	
PF_ATMPVC	ATM PVCs	

PF_X25	(Reserved for X.25 project)
	
	#include <linux/x25.h>
		typedef struct {
		    char x25_addr[16];
		} x25_address;
		
		struct sockaddr_x25 {
		    sa_family_t sx25_family;
		/* X.121 Address */
		    x25_address sx25_addr;
		};

PF_INET6	IPv6 protocol family
	
	#include <linux/in6.h>
		struct in6_addr
		{
		    union {
		        u8 u6_addr8[16];
		      __u16 u6_addr16[8];
		      __u32 u6_addr32[4];
		#if (~0UL) > 0xffffffff
		#ifndef __RELAX_IN6_ADDR_ALIGNMENT
		/* Alas, protocols do not respect
		64bit alignment. rsvp/pim/... are
		broken. However, it is good idea to
		force correct alignment, when
		  it is possible. */
		     __u64 u6_addr64[2];
		#endif
		#endif

TABLE A.1 Continued

Type	*Description and Example*	*Affiliated Data*

		`    } in6_u;`
		`#define s6_addr   in6_u.u6_addr8`
		`#define s6_addr16 in6_u.u6_addr16`
		`#define s6_addr32 in6_u.u6_addr32`
		`#define s6_addr64 in6_u.u6_addr64`
		`};`
		
		`struct sockaddr_in6 {`
		`    unsigned short int sin6_family;`
		`  __u16               sin6_port;`
		`  __u32               sin6_flowinfo;`
		`    struct in6_addr    sin6_addr;`
		`};`

`PF_ROSE`	Amateur Radio X.25 PLP
	
	`#include <linux/rose.h>`
	`struct sockaddr_rose {`
	`    sa_family_t  srose_family;`
	`    rose_address srose_addr;`
	`    ax25_address srose_call;`
	`    int          srose_ndigis;`
	`    ax25_address srose_digi;`
	`};`
	
	`struct full_sockaddr_rose {`
	`    sa_family_t  srose_family;`
	`    rose_address srose_addr;`
	`    ax25_address srose_call;`
		`typedef struct {`
		`    char rose_addr[5];`
		`} rose_address;`

TABLE A.1 Continued

Type	*Description and Example*	*Affiliated Data*

		`unsigned int srose_ndigis;`
		`ax25_address`
		`srose_digis[ROSE_MAX_DIGIS];`
		`};`

`PF_DECnet`	(Reserved for DECnet project)	

`PF_NETBEUI`	(Reserved for 802.2LLC project)
	
	`#include <linux/netbeui.h>`
		`#define NB_NAME_LEN 20`
		
		`struct sockaddr_netbeui`
		`{`
		`sa_family snb_family;`
		`char snb_name[NB_NAME_LEN];`
		`char snb_devhint[IFNAMSIZ];`
		`};`

`PF_SECURITY`	Security callback pseudo AF	
`PF_KEY`	`PF_KEY` key management API	

`PF_NETLINK`
`PF_ROUTE`
	Aliases to emulate 4.4 BSD
	
	`#include <linux/netlink.h>`
		`struct sockaddr_nl`
		`{`
		`sa_family_t nl_family;`
		`unsigned short nl_pad;`
		`__u32 nl_pid;`
		`__u32 nl_groups;`
		`};`

`PF_PACKET`	Packet family	`struct sockaddr_pkt` `{`

TABLE A.1 Continued

Type	*Description and Example*	*Affiliated Data*

	`#include <linux/if_packet.h>`	`unsigned short spkt_family;`
		`unsigned char spkt_device[14];`
		`unsigned short spkt_protocol;`
		`};`
		
		`struct sockaddr_ll`
		`{`
		`unsigned short sll_family;v`
		`unsigned short sll_protocol;`
		`int sll_ifindex;`
		`unsigned short sll_hatype;`
		`unsigned char sll_pkttype;`
		`unsigned char sll_halen;`
		`unsigned char sll_addr[8];`
		`};`

`PF_ASH`	Ash	

`PF_ECONET`	Acorn Econet
	
	`#include <linux/if_ec.h>`
		`struct ec_addr`
		`{`
		`/* Station number. */`
		`unsigned char station;`
		`/* Network number. */`
		`unsigned char net;`
		`};`
		
		`struct sockaddr_ec`
		`{`
		`unsigned short sec_family;`
		`unsigned char port;`

TABLE A.1 Continued

Type	*Description and Example*	*Affiliated Data*

		`/* Control/flag byte.  */`
		`  unsigned char cb;`
		`/* Type of message.  */`
		`  unsigned char type;`
		`  struct ec_addr addr;`
		`  unsigned long cookie;`
		`};`

`PF_ATMSVC`	ATM SVCs	
`PF_SNA`	Linux SNA project	

`PF_IRDA`	IRDA sockets
	
	`#include <linux/irda.h>`
		`struct sockaddr_irda {`
		`  sa_family_t  sir_family;`
		`/* LSAP/TSAP selector */`
		`  unsigned char sir_lsap_sel;`
		`/* Device address */`
		`  unsigned int  sir_addr;`
		`/* Usually <service>:IrDA:TinyTP */`
		`  char  sir_name[25];`
		`};`

Types: Second Parameter to `socket()`

The second parameter (`type`) selects the protocol layer. Some constants defined in Table A.2 are mere placeholders for when the kernel supports the protocol.

TABLE A.2 Protocol Values for the `type` Parameter in `socket()` Call

Protocol Type	*Description*
`SOCK_STREAM`	(TCP) Reliable, two-way communication in a stream. You can use this kind of socket in higher-level I/O function calls that involve the `FILE*` type. This protocol offers you a virtual connection to the network using ports and a dedicated client channel. After connection, the `accept()` call returns a new socket descriptor specifically for the new client.
`SOCK_DGRAM`	(UDP) Unreliable connectionless communication. Each message is independent and can be lost during transmission. This protocol virtualizes the network with ports and allows you to send and receive messages from many peers without reconnection.
`SOCK_RAW`	(IP) Accesses the internal network interfaces and fields. If you want to create ICMP messages, you need to create a raw socket. Root access only.
`SOCK_RDM`	(RDM—Reliably Delivered Messages) Makes sure that each packet arrives safely to the destination, but does not guarantee correct packet order. (Not yet implemented in Linux and other UNIX operating systems.)
`SOCK_SEQPACKET`	Sequenced, reliable, connection-based datagrams of fixed length. (Not yet implemented in Linux.)
`SOCK_PACKET`	(Physical Layer) Places the socket in promiscuous mode (if available) where it will receive any and all packets on the network. This is a Linux-only tool. Root-access only. (Deprecated—use `PF_PACKET` instead.)

Protocol Definitions

Listing A.1 is an excerpt from the `/etc/protocols` file [RFC2292] on your distribution. It contains the common and standard protocol standards used in the network packet. Revising this file is *not* a good idea.

LISTING A.1 `/etc/protocols` File

```
ip          0 IP         # internet protocol, pseudo number
icmp        1 ICMP       # internet control message protocol
igmp        2 IGMP       # Internet Group Management
ggp         3 GGP        # gateway-gateway protocol
ipencap     4 IP-ENCAP   # IP encapsulated in IP
st          5 ST         # ST datagram mode
tcp         6 TCP        # transmission control protocol
egp         8 EGP        # exterior gateway protocol
pup        12 PUP        # PARC universal packet protocol
udp        17 UDP        # user datagram protocol
hmp        20 HMP        # host monitoring protocol
xns-idp    22 XNS-IDP    # Xerox NS IDP
rdp        27 RDP        # "reliable datagram" protocol
iso-tp4    29 ISO-TP4    # ISO Transport Protocol class 4
xtp        36 XTP        # Xpress Tranfer Protocol
ddp        37 DDP        # Datagram Delivery Protocol
idpr-cmtp  39 IDPR-CMTP  # IDPR Control Message Transport
rspf       73 RSPF       # Radio Shortest Path First
vmtp       81 VMTP       # Versatile Message Transport
ospf       89 OSPFIGP    # Open Shortest Path First IGP
ipip       94 IPIP       # Yet Another IP encapsulation
encap      98 ENCAP      # Yet Another IP encapsulation
```

Standard Internet Port Assignments (First 100 Ports)

Listing A.2 shows the standard ports (up to port #100) defined in the `/etc/services` file. You can change many of these to suit your needs, but be sure to notify the clients if you do so.

LISTING A.2 `/etc/services` File

```
tcpmux      1/tcp                    # TCP port service multiplexer
rtmp        1/ddp                    # Routing Table Maintenance Protocol
nbp         2/ddp                    # Name Binding Protocol
echo        4/ddp                    # AppleTalk Echo Protocol
zip         6/ddp                    # Zone Information Protocol
echo        7/tcp
echo        7/udp
discard     9/tcp     sink null
discard     9/udp     sink null
systat      11/tcp    users
daytime     13/tcp
```

LISTING A.2 Continued

```
daytime     13/udp
netstat     15/tcp
qotd        17/tcp   quote
msp         18/tcp                  # message send protocol
msp         18/udp                  # message send protocol
chargen     19/tcp   ttytst source
chargen     19/udp   ttytst source
ftp-data    20/tcp
ftp         21/tcp
fsp         21/udp   fspd
ssh         22/tcp                  # SSH Remote Login Protocol
ssh         22/udp                  # SSH Remote Login Protocol
telnet      23/tcp
# 24 - private
smtp        25/tcp   mail
# 26 - unassigned
time        37/tcp   timserver
time        37/udp   timserver
rlp         39/udp   resource        # resource location
nameserver  42/tcp   name            # IEN 116
whois       43/tcp   nicname
re-mail-ck  50/tcp                  # Remote Mail Checking Protocol
re-mail-ck  50/udp                  # Remote Mail Checking Protocol
domain      53/tcp   nameserver      # name-domain server
domain      53/udp   nameserver
mtp         57/tcp                  # deprecated
bootps      67/tcp                  # BOOTP server
bootps      67/udp
bootpc      68/tcp                  # BOOTP client
bootpc      68/udp
tftp        69/udp
gopher      70/tcp                  # Internet Gopher
gopher      70/udp
rje         77/tcp   netrjs
finger      79/tcp
www         80/tcp   http            # WorldWideWeb HTTP
www         80/udp                  # HyperText Transfer Protocol
link        87/tcp   ttylink
kerberos    88/tcp   kerberos5 krb5  # Kerberos v5
kerberos    88/udp   kerberos5 krb5  # Kerberos v5
supdup      95/tcp
linuxconf   98/tcp
# 100 - reserved
```

HTTP 1.1 Status Codes

If you want to write your own Web server, you need to understand and use the standard status codes of HTTP 1.1 [RFC2616, RFC2817]. Table A.3 lists these codes.

TABLE A.3 HTTP Result Codes

Code Class Value	*Class Name*	*Specific Code and Description*
1xx	Informational	100 Continue
		101 Switching Protocols
2xx	Successful	200 OK
		201 Created
		202 Accepted
		203 Non-Authoritative Information
		204 No Content
		205 Reset Content
		206 Partial Content
3xx	Redirection	300 Multiple Choices
		301 Moved Permanently
		302 Moved Temporarily
		303 See Other
		304 Not Modified
		305 Use Proxy
4xx	Client Error	400 Bad Request
		401 Unauthorized
		402 Payment Required
		403 Forbidden
		404 Not Found
		405 Method Not Allowed
		406 Not Acceptable
		407 Proxy Authentication Required
		408 Request Timeout
		409 Conflict
		410 Gone
		411 Length Required

TABLE A.3 Continued

Code Class Value	*Class Name*	*Specific Code and Description*
		412 Precondition Failed
		413 Request Entity Too Large
		414 Request-URI Too Long
		415 Unsupported Media Type
5xx	Server Error	500 Internal Server Error
		501 Not Implemented
		502 Bad Gateway
		503 Service Unavailable
		504 Gateway Timeout
		505 HTTP Version Not Supported

Socket Options (`get/setsockopt()`)

Tables A.4 through A.7 list the various socket options and the required parameters. Not all options are size compatible between UNIX types. For example, `IP_TTL` in Linux allows a type `int`, but only fills the first byte. IBM's AIX restricts the same option to a `char` instead.

TABLE A.4 General Socket Options

Level	*Option*	*Description*	*	*R*	*W*	*Value*	*Type*
`SOL_SOCKET`	`SO_ATTACH_FILTER`	Attach filter	?	?	?	Integer	`int`
`SOL_SOCKET`	`SO_BINDTODEVICE`	Bind to device	?	?	?	String	`char*`
`SOL_SOCKET`	`SO_BROADCAST`	Enable broadcast	Y	Y	Y	Boolean	`int`
`SOL_SOCKET`	`SO_BSDCOMPAT`	Request BSD bug-for-bug compatibility	Y	Y	Y	Boolean	`int`
`SOL_SOCKET`	`SO_DEBUG`	Enable socket debugging	Y	Y	Y	Boolean	`int`
`SOL_SOCKET`	`SO_DETACH_FILTER`	Detach filter	?	?	?	Integer	`int`
`SOL_SOCKET`	`SO_DONTROUTE`	Forbid routing	Y	Y	Y	Boolean	`int`
`SOL_SOCKET`	`SO_ERROR`	Last error	Y	Y	Y	Integer	`int`

TABLE A.4 Continued

Level	*Option*	*Description*	*	*R*	*W*	*Value*	*Type*
SOL_SOCKET	SO_KEEPALIVE	Enable keeping connection alive	Y	Y	Y	Boolean	int
SOL_SOCKET	SO_LINGER	Linger until data sent	Y	Y	Y	Linger	struct linger
SOL_SOCKET	SO_NO_CHECK	No checking	Y	Y	Y	Boolean	int
SOL_SOCKET	SO_OOBINLINE	Place out-of-band inline	Y	Y	Y	Boolean	int
SOL_SOCKET	SO_PASSCRED	Enable passing user credentials	Y	Y	Y	Boolean	int
SOL_SOCKET	SO_PEERCRED	Peer Credentials	Y	Y	Y	Credentials	struct ucred
SOL_SOCKET	SO_PRIORITY	Set the queue priority	Y	Y	Y	Integer	int
SOL_SOCKET	SO_RCVBUF	Receive buffer size	Y	Y	Y	Integer	int
SOL_SOCKET	SO_RCVLOWAT	Receive low water mark	Y	Y	N	Integer	int
SOL_SOCKET	SO_RCVTIMEO	Receive Timeout	Y	Y	Y	Time	struct timeval
SOL_SOCKET	SO_REUSEADDR	Reuse Address	Y	Y	Y	Boolean	int
SOL_SOCKET	SO_REUSEPORT	Reuse Address (multicasting)	N	-	-	Boolean	int
SOL_SOCKET	SO_SECURITY_AUTHENTICATION	Security authentication	N	-	-	Integer	int
SOL_SOCKET	SO_SECURITY_ENCRYPTION_NETWORK	Security encryption network	N	-	-	Integer	int
SOL_SOCKET	SO_SECURITY_ENCRYPTION_TRANSPORT	Security encryption transport	N	-	-	Integer	int
SOL_SOCKET	SO_SNDBUF	Send buffer size	Y	Y	Y	Integer	int
SOL_SOCKET	SO_SNDLOWAT	Send low water mark	Y	Y	Y	Integer	int
SOL_SOCKET	SO_SNDTIMEO	Send Timeout	Y	Y	Y	Time	struct timeval
SOL_SOCKET	SO_TYPE	Socket Type	Y	Y	Y	Integer	int

TABLE A.5 IP-Level Socket Options

Level	*Option*	*Description*	*	*R*	*W*	*Value*	*Type*
SOL_IP	IP_ADD_MEMBERSHIP	Add multicast membership	Y	Y	Y	IPv4 Multicast Address	struct ip_mreq
SOL_IP	IP_DROP_MEMBERSHIP	Drop multicast membership	Y	Y	Y	IPv4 Multicast Address	struct ip_mreq
SOL_IP	IP_HDRINCL	Enable manual IP header creation	Y	Y	Y	Boolean	int
SOL_IP	IP_MTU_DISCOVER	MTU discover	Y	Y	Y	Integer	int
SOL_IP	IP_MULTICAST_IF	Outgoing multicast interface	Y	Y	Y	IPv4 Address	struct in_addr
SOL_IP	IP_MULTICAST_LOOP	Enable multicast loopback	Y	Y	Y	Boolean	int
SOL_IP	IP_MULTICAST_TTL	Multicast TTL	Y	Y	Y	Integer	int
SOL_IP	IP_OPTIONS	IP options	Y	Y	Y	Options	int[ ]
SOL_IP	IP_PKTINFO	Enable getting packet info	Y	Y	Y	Boolean	int
SOL_IP	IP_PKTOPTIONS	Packet options	N	-	-	Options	int[ ]
SOL_IP	IP_RECVERR	Enable receive error packets	Y	Y	Y	Boolean	int
SOL_IP	IP_RECVOPTS	Enable receive options	Y	Y	Y	Boolean	int
SOL_IP	IP_RECVTOS	Get received TOS	Y	Y	Y	Integer	int

TABLE A.5 Continued

Level	*Option*	*Description*	*	*R*	*W*	*Value*	*Type*
SOL_IP	IP_RECVTTL	Get received TTL	Y	Y	Y	Integer	int
SOL_IP	IP_RETOPTS	RETOPTS	Y	Y	Y	Boolean	int
SOL_IP	IP_ROUTER_ALERT	Enable router alerts	N	-	-	Boolean	int
SOL_IP	IP_TOS	Type of Service (TOS)	Y	Y	Y	Integer	int
SOL_IP	IP_TTL	Time to Live (TTL)	Y	Y	Y	Integer	int

TABLE A.6 IPv6-Level Socket Options

Level	*Option*	*Description*	*	*R*	*W*	*Value*	*Type*
SOL_IPV6	IPV6_ADD_MEMBERSHIP	Join multicast membership	?	?	?	IPv6 Multicast Address	struct ip_mreq6
SOL_IPV6	IPV6_ADDRFORM	Change address for socket	?	?	?	Integer	int
SOL_IPV6	IPV6_AUTHHDR	AUTHHDR	?	?	?	Integer	int
SOL_IPV6	IPV6_CHECKSUM	Offset of checksum for raw sockets	?	?	?	Integer	int
SOL_IPV6	IPV6_DROP_MEMBERSHIP	Drop multicast membership	?	?	?	IPv6 Multicast Address	struct ip_mreq6

TABLE A.6 Continued

Level	*Option*	*Description*	*	*R*	*W*	*Value*	*Type*
SOL_IPV6	IPV6_DSTOPTS	Enable getting destination options	?	?	?	Boolean	int
SOL_IPV6	IPV6_HOPLIMIT	Enable getting hop limit	?	?	?	Boolean	int
SOL_IPV6	IPV6_HOPOPTS	Enable getting hop-by-hop options	?	?	?	Boolean	int
SOL_IPV6	IPV6_MULTICAST_HOPS	Specify number of multicast hops	?	?	?	Integer	int
SOL_IPV6	IPV6_MULTICAST_IF	Specify outgoing multicast interface	?	?	?	IPv6 Address	struct in6_addr
SOL_IPV6	IPV6_MULTICAST_LOOP	Enable multicast loopback	?	?	?	Integer	int
SOL_IPV6	IPV6_NEXTHOP	Enable specifying next hop	?	?	?	Boolean	int
SOL_IPV6	IPV6_PKTINFO	Receive packet information	?	?	?	Integer	int
SOL_IPV6	IPV6_PKTOPTIONS	Specify packet options	?	?	?	Options	int []
SOL_IPV6	IPV6_ROUTER_ALERT	Enable router alerts	?	?	?	Boolean	int
SOL_IPV6	IPV6_RXSRCRT	Receive source route	?	?	?	Boolean	int
SOL_IPV6	IPV6_UNICAST_HOPS	Specify hop limit	?	?	?	Integer	int

TABLE A.7 TCP-Level Socket Options

Level	*Option*	*Description*	*	*R*	*W*	*Value*	*Type*
SOL_TCP	TCP_KEEPALIVE	Keep alive delay (replaced with sysctl call)	N	-	-	Integer	int
SOL_TCP	TCP_MAXRT	Maximum retransmit time	N	-	-	Integer	int
SOL_TCP	TCP_MAXSEG	Set max segment (transmission buffer) size	Y	Y	Y	Integer	int
SOL_TCP	TCP_NODELAY	Enable Nagle algorithm	Y	Y	Y	Boolean	int
SOL_TCP	TCP_STDURG	Specifies location of urgent byte (replaced with sysctl call)	N	-	-	Boolean	int
SOL_TCP	TCP_CORK	Never send partially complete segments	Y	Y	Y	Boolean	int
SOL_TCP	TCP_KEEPIDLE	Start keep-alives after this period	Y	Y	Y		int
SOL_TCP	TCP_KEEPINTVL	Interval between keep-alives	Y	Y	Y		int
SOL_TCP	TCP_KEEPCNT	Number of keep-alives before death	Y	Y	Y		int
SOL_TCP	TCP_SYNCNT	Number of SYN retransmits	Y	Y	Y		int
SOL_TCP	TCP_LINGER2	Lifetime of orphaned FIN-WAIT-2 state	Y	Y	Y		int
SOL_TCP	TCP_DEFER_ ACCEPT	Wake up listener only when data arrive	Y	Y	Y		int
SOL_TCP	TCP_WINDOW_ CLAMP	Bound advertised window	Y	Y	Y		int

Signal Definitions

Table A.8 lists the standard signals and their meanings.

TABLE A.8 Standard Linux Signal Codes

Signal	*Value*	*Action*	*Comment*
SIGHUP	1	A	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	A	Interrupt from keyboard
SIGQUIT	3	A	Quit from keyboard
SIGILL	4	A	Illegal instruction
SIGABRT	6	C	Abort signal from `abort(3)`
SIGFPE	8	C	Floating point exception
SIGKILL	9	AEF	Kill signal
SIGSEGV	11	C	Invalid memory reference
SIGPIPE	13	A	Broken pipe: write to pipe with no readers
SIGALRM	14	A	Timer signal from `alarm(2)`
SIGTERM	15	A	Termination signal
SIGUSR1	30,10,16	A	User-defined signal 1
SIGUSR2	31,12,17	A	User-defined signal 2
SIGCHLD	20,17,18	B	Child stopped or terminated
SIGCONT	19,18,25		Continue if stopped
SIGSTOP	17,19,23	DEF	Stop process
SIGTSTP	18,20,24	D	Stop typed at tty
SIGTTIN	21,21,26	D	tty input for background process
SIGTTOU	22,22,27	D	tty output for background process
SIGIOT	6	CG	IOT trap; A synonym for SIGABRT
SIGEMT	7,-,7	G	
SIGBUS	10,7,10	AG	Bus error
SIGSYS	12,-,12	G	Bad argument to routine (SVID)
SIGSTKFLT	-,16,-	AG	Stack fault on coprocessor
SIGURG	16,23,21	BG	Urgent condition on socket (4.2 BSD)
SIGIO	23,29,22	AG	I/O now possible (4.2 BSD)
SIGPOLL		AG	A synonym for SIGIO (System V)

TABLE A.8 Continued

Signal	*Value*	*Action*	*Comment*
SIGCLD	-,-,18	G	A synonym for SIGCHLD
SIGXCPU	24,24,30	AG	CPU time limit exceeded (4.2 BSD)
SIGXFSZ	25,25,31	AG	File size limit exceeded (4.2 BSD)
SIGVTALRM	26,26,28	AG	Virtual alarm clock (4.2 BSD)
SIGPROF	27,27,29	AG	Profile alarm clock
SIGPWR	29,30,19	AG	Power failure (System V)
SIGINFO	29,-,-	G	A synonym for SIGPWR
SIGLOST	-,-,-	AG	File lock lost
SIGWINCH	28,28,20	BG	Window resize signal (4.3 BSD, Sun)
SIGUNUSED	-,31,-	AG	Unused signal

A—Default action is to terminate the process.
B—Default action is to ignore the signal.
C—Default action is to dump core.
D—Default action is to stop the process.
E—Signal cannot be caught.
F—Signal cannot be ignored.
G—Not a POSIX.1-conformant signal.

ICMP Codes

Table A.9 shows the different types of ICMP [RFC792] packets and what they mean.

TABLE A.9 ICMP Code Descriptions

Type	*Code*	*Description*
0	0	Echo reply
3		Destination unreachable
	0	Network unreachable
	1	Host unreachable
	2	Protocol unreachable
	3	Port unreachable
	4	Fragmentation needed but DF bit set
	5	Source route failed
	6	Destination network unknown

TABLE A.9 Continued

Type	*Code*	*Description*
	7	Destination host unknown
	8	Source host isolated (obsolete)
	9	Destination network administratively prohibited
	10	Destination host administratively prohibited
	11	Network unreachable to TOS
	12	Host unreachable for TOS
	13	Communication administratively prohibited
	14	Host precedence violation
	15	Precedence cutoff in effect
4	0	Source quench—gateway requesting the host to reduce transfer rate
5		Redirect
	0	Redirect for network
	1	Redirect for host
	2	Redirect for type-of-service and network
	3	Redirect for type-of-service and host
8	0	Echo request
9	0	Router advertisement
10	0	Router solicitation
11		Time exceeded
	0	TTL equals 0 during transit
	1	TTL equals 0 during fragment reassembly
12		Parameter problem
	0	IP header bad
	1	Required option missing
13	0	Timestamp request
14	0	Timestamp reply
15	0	Information request
16	0	Information reply
17	0	Address mask request
18	0	Address mask reply

IPv4 Multicast Allocation

Table A.10 defines the current multicast address allocation [RFC2365] in spectrum order.

TABLE A.10 Proposed Multicast Allocation

Address Range	*Scope*	*Typical TTL*
`224.0.0.0`–`224.0.0.255`	Cluster	0
`224.0.1.0`–`238.255.255.255`	Global	<= 255
`239.0.0.0`–`239.191.255.255`	(undefined)	
`239.192.0.0`–`239.195.255.255`	Organization	< 128
`239.196.0.0`–`239.254.255.255`	(undefined)	
`239.255.0.0`–`239.255.255.255`	Site	< 32

Proposed IPv6 Address Allocation

Table A.11 lists the proposed IPv6 address allocation in bits.

TABLE A.11 Proposed IPv6 Address Allocation

Allocation	*Address Prefix*
(reserved)	0000 0000
(unassigned)	0000 0001
NSAP	0000 001
IPX	0000 010
(unassigned)	0000 011
(unassigned)	0000 1
(unassigned)	0001
Aggregate global unicast addresses	001
(unassigned)	010
(unassigned)	011
(unassigned)	100
(unassigned)	101
(unassigned)	110
(unassigned)	1110
(unassigned)	1111 0

TABLE A.11 Continued

Allocation	*Address Prefix*
(unassigned)	1111 10
(unassigned)	1111 110
(unassigned)	1111 1110 0
Link-local unicast address	1111 1110 10
Site-local unicast address	1111 1110 11
Multicast address	1111 1111

ICMPv6 Codes

Table A.12 shows the new ICMPv6 [RFC2463] for IPv6.

TABLE A.12 ICMPv6 Code Descriptions

Type	*Code*	*Description*
1		Destination unreachable
	0	No route to destination
	1	Administratively prohibited (firewall filter)
	2	Not a neighbor (incorrect strict source route)
	3	Address unreachable (general)
	4	Port unreachable
2	0	Packet too big
3		Time exceeded
	0	Hop limit exceeded during transit
	1	Fragment reassembly time exceeded
4		Parameter problem
	0	Erroneous header field
	1	Unrecognized next header
	2	Unrecognized option
128	0	Echo request (ping)
129	0	Echo reply (ping)
130	0	Group membership query
131	0	Group membership report

TABLE A.12 Continued

Type	*Code*	*Description*
132	0	Group membership reduction
133	0	Router solicitation
134	0	Router advertisement
135	0	Neighbor solicitation
136	0	Neighbor advertisement
137	0	Redirect

IPv6 Multicast Scope Field

Table A.13 defines the different values for the scope field in the multicast IPv6 addresses.

TABLE A.13 IPv6 Multicast Scope Field Descriptions

Scope	*Range*	*Description*
0	(undefined)	
1	Node	Local on the same host (like `127.0.0.1`).
2	Link	Messages stay within the router's group. The routers never let these messages pass through.
3–4	(undefined)	
5	Site	As defined by the network administrators, the messages stay within the site's locality.
6–7	(undefined)	
8	Organization	As defined by the network administrators, the messages stay within the organization's locality.
9–13	(reserved)	
14	Global	All routers permit messages to pass through the global network until they expire.
15	(reserved)	

IPv6 Multicast Flags Field

Table A.14 shows the currently defined flag fields for IPv6 multicast addresses.

TABLE A.14 IPv6 Multicast Flags Field Descriptions

Bit #	*Description*
0	Transience
	0= well known address
	1= transient address
1	(reserved)
2	(reserved)
3	(reserved)