STREAMS Data Structures A

This appendix summarizes data structures commonly encountered in STREAMS module and driver development. Most of the data structures given in this appendix are contained in <sys/stream.h> and are documented in the man pages.

Many of the fields in the structures described below are intended for the private use of the STREAMS framework code. You should not permit modules and drivers to access these fields in any way, as their meaning, existence and size may change from release to release. *These fields may be omitted from the following descriptions.*

streamtab

This structure defines a module or a driver.

```
struct streamtab {
    struct qinit             *st_rdinit;   /* defines read queue */
    struct qinit             *st_wrinit;   /* defines write queue */
    struct qinit             *st_muxrinit; /* for multiplexing */
    struct qinit             *st_muxwinit; /* drivers only */
};
```

QUEUE Structures

Two sets of queue structures form a module. The structures are `queue`, `qinit`, `module_info`, and `module_stat` (optional).

queue

The `queue`(9S) structure has the following format:

```
struct qinit       *q_qinfo; /* procs and limits for queue */
struct msgb        *q_first; /* msg que head for this queue */
struct msgb        *q_last;  /* msg queue tail for this queue */
struct queue       *q_next;  /* next queue in Stream */
struct queue       *q_link   /* to next Q for scheduling */
void            *q_ptr; /* to private data structure */
ulong               q_count;  /* number of bytes in queue */
ulong               q_flag;   /* queue state */
long                q_minpsz; /* min packet size accepted */
long                q_maxpsz; /* max packet size accepted */
ulong               q_hiwat;  /* queue high water mark */
ulong               q_lowat;  /* queue low water mark */
```

When a `queue` pair is allocated, their contents are zero unless specifically initialized. The following fields are initialized:

- `q_qinfo`: `st_rdinit` and `st_wrinit` (or `st_muxrinit` and `st_muxwinit`) - from `streamtab`
- `q_minpsz, q_maxpsz, q_hiwat, q_lowat` - from `module_info`
- `q_ptr` - optionally, by the driver/module open routine

Queue flags from `queue(9S)` for queue structure are defined as:

```
#define QENAB                  0x001      /* Queue is already enabled to run */
#define QWANTR                 0x002      /* Someone wants to read Q */
#define QWANTW                 0x004      /* Someone wants to write Q */
#define QFULL                  0x008      /* Q is considered full */
#define QREADR                 0x010      /* This is the reader (first) Q */
#define QUSE                   0x020      /* This queue in use (allocation) */
```

```
#define QNOENB              0x040      /* Don't enable Q via putq */
#define QOLD                0x080      /* Pre-SVR4 open/close interface */
#define QBACK               0x100      /* queue has been back-enabled */
```

qinit

`qinit`(9S) format is as follows:

```
struct qinit {
    int                  (*qi_putp)();        /* put procedure */
    int                  (*qi_srvp)();        /* service procedure */
    int                  (*qi_qopen)();       /*called on each open or push*/
    int                  (*qi_qclose)();      /*called on last close or pop*/
    int                  (*qi_qadmin)();      /* reserved for future use */
    struct module_info *qi_minfo;            /* info struct */
    struct module_stat *qi_mstat;            /*stats struct (opt)*/
};
```

module_info

`module_info` (9S) has the following format:

```
struct module_info {
    ushort     mi_idnum;                 /* module ID number */
    char       *mi_idname;               /* module name */
    long       mi_minpsz;                /* min packet size accepted */
    long       mi_maxpsz;                /* max packet size accepted */
    ulong      mi_hiwat;                 /* high water mark, flow ctrl */
    ulong      mi_lowat;                 /* low water mark, flow ctrl */
};
```

qband

The queue flow information, `qband` (9S) for each band is contained in the following structure:

```
/* Structure that describes the separate information
 * for each priority band in the queue
 */
struct qband {
    struct qband           *qb_next;         /* next band's info */
    ulong                  qb_count;         /* number of bytes in band */
    struct msgb            *qb_first;        /* beginning of band's data */
    struct msgb            *qb_last;         /* end of band's data */
    ulong                  qb_hiwat;         /* high water mark for band */
    ulong                  qb_lowat;         /* low water mark for band */
    ulong                  qb_flag;          /* QB_FULL, denotes that a
                                             band of data flow is flow
                                             controlled */
};
```

Message Structures

A message is composed of a linked list of triples, consisting of two structures (`msgb`(9S) and `datab`(9S)) and a data buffer.

```
struct msgb {
    struct msgb            *b_next;      /*next msg on queue*/
    struct msgb            *b_prev;      /*previous msg on queue*/
    struct msgb            *b_cont;      /*next msg block of message*/
    unsigned char          *b_rptr;      /*1st unread byte in bufr*/
    unsigned char          *b_wptr;      /*1st unwritten byte in bufr*/
    struct datab           *b_datap;     /*data block*/
    unsigned char          b_band;       /*message priority*/

    unsigned short         b_flag;       /*see below - Message flags*/
};typedef struct msgb mblk_t;
```

Note – Modules or drivers cannot modify `b_next` and `b_prev`. These fields are modified by utility routines such as `putq()` and `getq()`.

Conceptually the band belongs in the message block since it is associated with the message and not just with the data. However, the size of a message block is visible to modules and drivers, so the band is placed in the data block instead. Modules and drivers should have no knowledge of the size of the data block.

```
struct datab {
    unsigned char   *db_base;           /* first byte of buffer */
    unsigned char   *db_lim;            /* last byte+1 of buffer */
    unsigned char   db_ref;             /* msg count ptg to this blk */
    unsigned char   db_type;            /* msg type */
};
typedef struct datab dblk_t;
```

iocblk

This is contained in an M_IOCTL message block:

```
struct iocblk {
    int         ioc_cmd;            /* ioctl command type */
    cred_t      *ioc_cr;            /* full credentials */
    uint        ioc_id;             /* ioctl id */
    uint        ioc_count;          /* count of bytes in data field */
    int         ioc_error;          /* error code */
    int         ioc_rval;           /* return value */
};
```

copyreq

This is used in `M_COPYIN/M_COPYOUT` messages:

```
struct copyreq {
    int                 cq_cmd;             /* ioctl command (from ioc_cmd) */
    cred_t              *cq_cr;             /* full credentials */
```

```
    uint               cq_id;             /* ioctl id (from ioc_id) */
    caddr_t            cq_addr;           /* address to copy data to/from */
    uint               cq_size;           /* number of bytes to copy */
    int                cq_flag;           /* see below */
    mblk_t             *cq_private; /* private state information */
};
```

copyresp

This structure is used in `M_IOCDATA`:

```
struct copyresp {
    int                cp_cmd;        /* ioctl command (from ioc_cmd) */
    cred_t             *cp_cr;        /* full credentials */
    uint               cp_id;         /* ioctl id (from ioc_id) */
    caddr_t            cp_rval;       /* status of req; 0 for success
                                       non-zero for failure */
     mblk_t             *cp_private; /* private state info */
};
```

Other Structures

strioctl

This structure supplies user values as an argument to the `ioctl` call `I_STR` in streamio(7).

```
struct strioctl {
    int        ic_cmd;            /* downstream request */
    int        ic_timout;         /*timeout acknowledgment-ACK/NAK*/
    int        ic_len;            /* length of data argument */
    char       *ic_dp;            /* pointer to data argument */
};
```

linkblk

This structure is used in lower multiplexer drivers to indicate a link

```
struct linkblk {
queue_t        *l_qtop;     /* lowest level write queue for upper */
                           /* Stream, set to NULL for persist links */
queue_t        *l_qbot;    /* high level write q of lower Stream */
int            l_index;    /* system-unique index for lower Stream */
};
```

stroptions

This structure holds various values used by the STREAMS. system:

```
struct stroptions {
    ulong              so_flags;              /* options to set */
    short              so_readopt;            /* read option */
    ushort             so_wroff;              /* write offset */
    long               so_minpsz;             /* minimum read packet size */
    long               so_maxpsz;             /* maximum read packet size */
    ulong              so_hiwat;              /* read queue high water mark */
    ulong              so_lowat;              /* read queue low water mark */
    unsigned charso_band;                     /* band for water marks */
};
/* flags for Stream options set message */
#define SO_ALL                      0x003f        /* set all options */
#define SO_READOPT                  0x0001        /* set read option */
#define SO_WROFF                    0x0002        /* set write offset */
#define SO_MINPSZ                   0x0004        /* set minimum packet size */
#define SO_MAXPSZ                   0x0008        /* set maximum packet size */
#define SO_HIWAT                    0x0010        /* set high water mark */
#define SO_LOWAT                    0x0020        /* set low water mark */
#define SO_MREADON                  0x0040        /* set read notification on */
#define SO_MREADOFF                 0x0080        /* set read notification off
*/
#define SO_NDELON                   0x0100      /* old TTY semantics for NDELAY
*/
#define SO_NDELOFF                  0x0200       /* STREAMS semantics for NDELAY
*/
```

```
#define SO_ISTTY          0x0400     /* Stream acting as
terminal*/
#define SO_ISNTTY         0x0800     /* Stream not acting as term*/
#define SO_TOSTOP         0x1000     /* stop on bkgrnd writes*/
#define SO_TONSTOP        0x2000     /* don’t stop on bkgrnd jobs*/
#define SO_BAND           0x4000     /* water marks affect band */
#define SO_DELIM          0x8000     /* messages are delimited */
#define SO_NODELIM        0x010000   /* turn off delimiters */
#define SO_STRHOLD        0x020000   /* strwrite msg coalescing */
```