```
	}

	sender = ad_sendr -> ad_r822adr;


	for (ap = ad_recip; ap; ap = ap -> ad_next) {
		for (up = arg ->users; up; up = up -> next) {
			if (up -> RecipientId -> parm != ap -> ad_no)
				continue;

			if (chan_acheck (ap, mychan,
					ad_list == NULL, &cur_mta) == NOTOK)
				continue;
		}
		if (up == NULL)
			continue;

		if (ad_list == NULLADDR)
			ad_list = alp = (ADDR *) calloc (1, sizeof *alp);
		else {
			alp -> ad_next = (ADDR *) calloc (1, sizeof *alp);
			alp = alp -> ad_next;
		}
		*alp = *ap;
		alp -> ad_next = NULLADDR;
	}

	if (ad_list == NULLADDR) {
		PP_LOG (LLOG_EXCEPTIONS, ("No recipients in user list"));
		rd_end ();
		return deliverystate;
	}

	deliver (ad_list, qp); /* do what is required */

	rd_end();

	return deliverystate;
}


static void dirinit()	/* Change into pp queue space */
{
	if (chdir (quedfldir) < 0)
		err_abrt (RP_LIO, "Unable to change directory to '%s'",
					quedfldir);
}
```

Figure 3.2: Example Outbound Channel

3.3.2 Address Parsing

Occasionally it is necessary for a channel to parse or manipulate an address. Most channels can get away without needing this however. Some of the basic parsing routines are given here.

The basic interface to parsing is `ad_parse`. This takes an address and attempts to parse it according to the given type and domain ordering. It is called as:

```
int ad_parse (ap, rp, order)
ADDR *ap;
RP_Buf *rp;
int order;
```

The parameters are:

`ap`: The address structure with the address to be parsed in the `ad_value` structure member. The address type should be set in `ad_type`.

`rp`: A reply structure which is updated with the results of the parsing.

`order`: The expected order of the domains. This is one of the following:

Value	Meaning
CH_USA_ONLY	US order domain only
CH_USA_PREF	US order prefered
CH_UK_ONLY	UK order domains only
CH_UK_PREF	UK order prefered

This is usually input from the channel structure.

This routine returns an `RP` code which can be checked with `rp_isgood` to see if the address was parsed. If it fails, a suitable error will be written in the `RP_Buf` structure.

When delivering to local addresses it is often necessary to reduce a global address down to a user of the system. The routine `ad_getlocal` can be of use here. This routine takes an address and returns just the local part of it if any, else `NULLCP`. It is called as follows:

```
char *ad_getlocal (addr, type, resp)
char *addr;
int type;
int resp;
```

The arguments are:

`addr`: A string-encoded form of the address; either an RFC 822 address or an RFC 1148 encoded X.400 address.

`type`: The type of address, either `AD_X400_type` or `AD_822_type`.

`resp`: If the address is a recipient, this value should be `1`, else `0`.

3.3.3 Local Pickup

This style of channel has not yet been written.

3.3.4 TWA Channel

This style of channel has not yet been written.

3.3.5 Delivery Reports

A typical interface to the delivery report functions is shown in Figure 3.3.

```
...
   {
           char buffer[BUFSIZ];

           (void) sprintf (buffer, "Address %s failed for reason %s",
                           ap -> ad_r822adr, rstr);

           (void) delivery_set (ap -> ad_no,
                           int_Qmgr_status_permanentFailure);
           set_dr (qp, ap -> ad_no, msg_name,
                   DRR_UNABLE_TO_TRANSFER,
                   DRD_UNRECOGNISED_OR, buffer);
   }
...
```

Figure 3.3: Example Delivery Report Generation

3.4 Formatters and Converters

Formatters and converters can run in two ways. Generally the simple filters work on single files, the complex filters work on several files at once. The majority of filters should be of the simple kind.

3.4.1 Simple Converters

The simple converters are programs that read from the standard input, do some formatting, and write the result to the standard output. This type of formatter is invoked by the filtercontrol program, `fcontrol`.

These simple filters should return a suitable exit status to confirm that the conversion completed successfully. They follow the standard UNIX[1] tradition of returning 0 on success and anything else on failure.

These filters may be anything from simple shell/awk/sed scripts to more complex *C* programs.

3.4.2 Complex Filters

More complex formatters are necessary if the formatting needs to convert more than one file. This is the case for formatters that change the directory structure, or combine several files into one. These formatters require more work, but are similar to the channel programs in their operation. The QMGR interface is in fact identical. The only real difference is in the `qid2dir` routine used to locate the directory. The starting directory is found using the normal `qid2dir` routine. The destination directory is found using the same routine, but first incrementing the formatter count in the address structure, and then calling the routine with the flag set to `FALSE`. The formatter should however place its output in a temporary directory which

[1] UNIX is a trademark of AT&T Bell Laboratories.

will then be renamed to the destination directory if the transformation is successful. This allows for failure at any point in the reformatting.

Index

A

addr/, 6
ADDR, 20
address structure, 20
addressing, 17
ad_getlocal, 44
ad_parse, 44
adr_add, 18
adr_free, 18
adr.h, 17
adr_init, 17
adr_new, 17
adr_tfree, 18
ASN.1, 5

B

/bin/mail, 28

C

chan_acheck, 40
chan_dcheck, 40
channel_control, 33
Chans/, 7
ch_mta2struct, 10
ch_nm2struct, 10

D

debugging, 30
delivery_init, 35
delivery_resetDRs, 35
delivery_set, 35
delivery_setall, 35
doc/, 7
dr.h, 38

F

fcontrol, 45
format/, 6

H

h/, 7

I

io/, 6
io_adend, 25
io_end, 26
io_init, 23
io_tdata, 26
io_tdend, 26
io_tend, 26
io_tinit, 25
io_tpart, 25
io_wadr, 25
io_wprm, 23
io_wrq, 25
isode/logger.h, 2

L

LaTeX, 47
lchan_acheck, 40
lconsole, 6
Lib/, 6
LINEconsole, 6
link, 26
list_bpt_add, 11
list_bpt_dup, 11
list_bpt_find, 11
list_bpt_free, 11
list_bpt_new, 10
LIST_RCHAN, 9
list_rchan_add, 10
list_rchan_free, 10
list_rchan_new, 9
list_rchan.h, 9
LLog, 2
ll_log.h, 3, 12
log_norm, 2
log_oper, 2
log_stat, 2

M

man/, 7
msg_rend, 37
msg_rfile, 37
msg_rinit, 37
MTAconsole/, 6

O

or/, 6

P

parameter structure, 11
parse/, 6
pp/, 6
PP_DBG, 5, 30
PP_DEBUG_ALL, 5
PP_DEBUG_SOME, 5
pp_initialise, 1
pp/Lib/tai/tai_sys.c, 2
PP_LOG, 3
PP_NOTICE, 4, 30
PP_OPER, 4
PP_PDU, 5
PP_PDUS, 30
pps_1adr, 29
pps_adr, 28
pps_aend, 29
pps_cc, 28
pps_end, 30
pps_file, 30
pps_hdr, 29
pps_init, 28
PP_SLOG, 4
PP_STAT, 4
pps_tinit, 29
pps_to, 28
pps_txt, 30
PP_TRACE, 5, 30
pptsapd/, 7
pptsapd, 33
prm, 11
prm_free, 11
prm.h, 11
prm_init, 11

Q

q_free, 14
q.h, 12
qid2dir, 37, 45
q_init, 14
qmgr/, 6, 7
Q_struct, 14, 25
queue structure, 14

R

rd_end, 36
rd_msg, 36
Reply structure, 22
retcode.h, 23
RP_Buf, 22, 23
rp_isbad, 23
rp_isgood, 23

S

set_1dr, 38
SLOG, 5
Src/, 6
status_failureSharedDR, 34
status_messageFailure, 34
status_mtaAndMessageFailure, 34
status_mtaFailure, 34
status_negativeDR, 34
status_positiveDR, 34
status_success, 34
status_successSharedDR, 34
struct prm_vars, 11
submit/, 7
submit, 8, 11, 12, 22, 23, 25, 26, 28
submit protocol, 8
sys_init, 1

T

table/, 6
tai/, 6
tailor, 6
Tools/, 7
type_Qmgr_Channel, 31
type_Qmgr_DeliveryStatus, 31
type_Qmgr_ProcMsg, 31
type_Qmgr_QID, 31
type_Qmgr_RecipientId, 31

U

Uip/, 7
util/, 6

W

wr_ad_rcntno, 36
wr_ad_status, 36
wr_q2dr, 39
wr_stat, 39

X

x400/, 6