/**

SPOOL
RP.LST
05/04/82
15:38:57

**/

```
SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE RP
OBJECT MODULE PLACED IN :F1:RP.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:RP.P86 OPTIMIZE(3) XREF SET(F1) DEBUG


              $TITLE('iLNA Transport Control Layer  Receive Process    04/15/82')
              $COMPACT DEBUG NOCOND
*** WARNING 10 IN 1 (LINE 2): RESPECIFIED PRIMARY CONTROL, IGNORED
              $SET(mipform)

              $IF f7
              $ELSE
              $INCLUDE (:F1:cpyrt.dcp)

          =           /* Intel Corporation Proprietary Information.
          =           This listing is supplied under the terms of a
          =           license agrement with Intel Corporaton and
          =           may not be copied nor disclosed except in
          =           accordance with the terms of that agreement. */

              $ENDIF


                        /*  George D Marshall   SC6-213   x7-5117  */

                       /* This is TCL's Receive Process (RP) and supporting routines.
                       RP's job is only to receive segments from Data LINK (or Network,
                       when there is one) and process them; only received segments
                       appear on RP's mailbox.  RPcauses segments to be transmitted by
                       updating shared variables in the connection data base (CDB), then
                       sending messages (Internal Request Blocks, or IRBs) to the Transmit
                       Process (TP) requesting transmission of a segment (such as SYN-ACK,
                       or ACK of a received data segment.  RP also clears and sets
                       the alarm control blocks in the CDB, which are events to TP when
                       and if they expire. */


                   /* Conditional assembly flags:
                       f7:     if true, all include files are taken from :F7:,
                                   if false, from :F1:.
                       log:    if true, code to handle trace buffers is included.
                       dbg:    if true, some additional debugging consistency
                                   checking code is included.
                       mipform: if true, link and blkptrs in RB are assumed to
                                   be in mipform, so they are converted to addresses.
                       */

                       /*
                         modified 03/10/ to change R. Shah's dynamic retransmission
                         policy to provide for less dynamic changes for dbp
                       */
1                 rp: DO;

              $IF f7
```

```
            $ELSE
            $INCLUDE (:F1:TCLGBL.INC)
        =                                                   /**********************/
        =                                                   /** Global Literals **/
        =                                                   /**********************/
        =   /* TCL Global Literals                                          04/15/82 */
  2  1  =   DECLARE
        =       max$send$seg          LITERALLY   '07H',  /* max no of back-to-back segs that  one connection */
        =                                                 /* can send at a time */
        =       tcl$header$len        LITERALLY   '20',   /* bytes in tcl header */

        =                                                 /* ETHERNET-SPECIFIC VALUES */
        =       dll$header$len        LITERALLY   '14',   /* bytes in dll header */
        =       min$pkt$len           LITERALLY   '46',   /* minimum total pkt len - bytes */
        =       max$seg$data$len$lit LITERALLY  '1480', /* (1480) max no. of client bytes in seg */
        =       tcl$protocol$code     LITERALLY   '5001H',/* DLLCONNECT user type field */
        =       tcl$protocol$code$rev LITERALLY '0150H',/* packet header user type field */

        =                                                 /* Misc values */
        =       tcl$mip$port          LITERALLY   '4',    /* mip port for IP$IN$MBX */
        =       log$rb$mip$port       LITERALLY   '5',    /* debugging: mip port for logging */
        =       mip$echo$port         LITERALLY   '7',    /* mip port of on-bd tcl echo server */

        =       tcl$version$lit       LITERALLY   '101H', /* Version of this TCL for seg header */
        =       def$net$id$lit        LITERALLY   '1',    /* default Network ID: "this network" */
        =       on$bd$tcl$echo$port LITERALLY   '7',    /* TCL port of on-board tcl echo server */
        =       true                  LITERALLY   'OFFH',
        =       false                 LITERALLY   '0',
        =       forever               LITERALLY   'WHILE true',
        =       Timeout$increase$state  LITERALLY '1',  /* In this state the retransmission timeout
        =                                                 is rapidly increased */
        =       Timeout$steady$state    LITERALLY '0';  /* In this state the timeout is
        =                                                 slowly decreased.  This should not be
        =                                                 changed, it is the initial state since
        =                                                 a cdb is intialised to zero */
            $ENDIF

            /* Some variables */
  3  1      DECLARE                                      /* note: externals are declared and documented
                                                          in TSTART */
              lcid$vector(*)        WORD    EXTERNAL,
              spec$type(*)          BYTE    EXTERNAL,
              num$cdbs              BYTE    EXTERNAL,
              cur$max$cdbs          BYTE    EXTERNAL,
              loc$net               WORD    EXTERNAL,
              loc$host(3)           WORD    EXTERNAL,
              tcl$version           WORD    EXTERNAL,
              min$retran$time       DWORD   EXTERNAL,
              Retran$increase       BYTE    EXTERNAL,
              cur$cdb$index         BYTE,
              cur$cid               WORD,
              cdbs$tried            BYTE,
              match$tries           BYTE,
              match$case            BYTE,
              rp$timestamp          DWORD,  /* temporary timestamp storage */
              rp$roundtrip          DWORD,
```

```
        (rp$roundtrip$lo,rp$roundtrip$hi)   WORD    AT(@rp$roundtrip),
        retran$weight       WORD    EXTERNAL,   /* weighting of old retransmit
                                                   timeout relative to just-computed
                                                   roundtrip, expressed as exponent of
                                                   two: 2^retran$weight is real weight */
        tot$pkts$rej        WORD    EXTERNAL,
        tot$pkts$retran     WORD    EXTERNAL,
        tot$rcv$buf$rej     WORD    EXTERNAL,
        bad$chk$sum         WORD    EXTERNAL,
        rp_has_ack          BYTE,           /* boolean for common test */
        rp_has_fin          BYTE,           /* boolean for common test */
        rp_ STRUCTURE(          /* copy of the segment header fields for code space*/
            dl$source0      WORD,
            dl$source1      WORD,
            dl$source2      WORD,
            dl$type         WORD,
            tcl$version     WORD,
            dest$port       WORD,
            source$port     WORD,
            dest$cid        WORD,
            source$cid      WORD,
            seg$seq$no      WORD,
            seg$ack$no      WORD,
            seg$data$len    WORD,
            ctl             WORD,       /* non-based copy of current rp.ctl */
            checksum        WORD)   PUBLIC, /* public for asm86 version of dlsource_eq_host */
$IF dbg
$ENDIF
        scratch             BYTE;
        $IF log
        $ENDIF

        $IF f7
        $ELSE
        $SAVE NOLIST INCLUDE(:F1:TCLMBX.INC)

5   1   DECLARE
            rp$p            POINTER,            /* Receive Packet */
            rp$o            WORD AT(@rp$p),
            rp BASED rp$o                       /* Based on offset portion for code */
                                                /* speed-up. Valid as long as segment */
                                                /* pool is in same Data Segment as */
                                                /* everything  else.  Probably will
                                                change for the CXU version */

        $IF f7
        $ELSE
        $INCLUDE(:F1:TCLSEG.INC)
=                                                       /**********************/
=                                                       /*** Segment Format **/
=                                                       /**********************/
=       STRUCTURE (                                        /*  11/15/81  */
=                               /* Offset (hex/ dec) */
=           kaos$msg$hdr POINTER,       /*  0 =  0T  ptr to link seg bufs */
=           buf$len     WORD,           /*  4 =  4T  # of bytes (dest addr thru last data byte) */
=                                                   /* DATA LINK FIELDS */
=           dl$dest(3)  WORD,           /*  6 =  6T  data link destination */
```

```
=           dl$source(3) WORD,          /*  C = 12T  data link source */
=           dl$type      WORD,          /* 12 = 18T  */
=                                           /* TRANSPORT CONTROL FIELDS */
=           tcl$version WORD,           /* 14 = 20t Version number of sending TCL */
=           dest$port   WORD,           /* 16 = 22T */
=           source$port WORD,           /* 18 = 24T */
=           dest$cid    WORD,           /* 1A = 26T */
=           source$cid  WORD,           /* 1C = 28T */
=           seg$seq$no  WORD,           /* 1E = 30T */
=           seg$ack$no  WORD,           /* 20 = 32T */
=           seg$data$len WORD,          /* 22 = 34T  # segment data bytes:for pad */
=           ctl          WORD,          /* 24 = 36T  subfields mapped out as:
=               (reserved)      bit,                8000H
=               cks             bit,                4000H
=               ayt             bit,                2000H
=               ack             bit,                1000H
=               syn             bit,                0800H
=               fin             bit,                0400H
=               eom             bit,                0200H
=               rst             bit,                0100H
=               (reserved)      bit,                0040H
=               credit        6 bits                003FH       */
=           checksum    WORD,           /* 26 = 38T */
=           seg$data(1) BYTE) /* 28 = 3AT segment data bytes, length unknown */
    $ENDIF
        ;

    $IF f7
    $ELSE
    $SAVE NOLIST INCLUDE (:F1:TCLSCF.INC)

7   1   DECLARE
        cur$cdb$p       POINTER,
                                                /**********************/
                                                /*** Conn Data Base **/
                                                /**********************/

        c BASED cur$cdb$p
    $IF f7
    $ELSE
    $SAVE NOLIST INCLUDE (:F1:TCLCDB.INC)
        ;
    $IF f7
    $ELSE
    $SAVE NOLIST INCLUDE (:F1:TCLCSD.INC)

9   1   DECLARE
        irb$p       POINTER,
        irb$o       WORD AT(@irb$p),
        irb BASED irb$o
    $IF f7
    $ELSE
    $INCLUDE (:F1:TCLIRB.INC)
=       STRUCTURE (   /* TCL Internal Request Block (IRB) format        03/01/81 */
=           cmx$ptr         POINTER,    /*  0       for CMX to link mbx buffers */
=           type            BYTE,       /*  4       request code (same position as alarm cb type field) */
=           cdb$index       BYTE,       /*  5       cdb index for above cid */
=           cid             WORD)       /*  6       CID for this irb */
```

```
=                                       /*  8       Total Length */
    $ENDIF
        ,
        lirb$p      POINTER,
        lirb$o      WORD AT(@lirb$p),
        lirb BASED lirb$o
    $IF f7
    $ELSE
    $INCLUDE (:F1:TLIRB.INC)
=                                                                   /*  03/21/81 */
=       STRUCTURE ( /* TCL Long Internal Request Block for RST segments. */
=                   /* this should always match the order of these fields in seg */
=           cmx$ptr             POINTER,/*          CMX field for mbx linkage */
=           type                BYTE,   /*  4       request code */
=           reason              BYTE,   /*  5       reason for the RST */
=           dl$dest (3)         WORD,   /*  6       dest host ID for the RST seg */
=           dest$port           WORD,   /*  C = 12t destination port for RST seg */
=           source$port         WORD,   /*  E = 14  source port for RST seg */
=           dest$cid            WORD,   /* 10 = 16  dest cid for RST seg */
=           source$cid          WORD,   /* 12 = 18  source cid for RST segment */
=           seg$seq$no          WORD,   /* 14 = 20  seq no to go in RST segment */
=           seg$ack$no          WORD)   /* 16   22  */
=                                       /* 18 = 24t Total Length */
    $ENDIF
        ;
    $IF f7
    $ELSE
    $INCLUDE (:F1:TCLIRC.INC)
=                                                       /**********************/
=                                                       /****  IRB  Codes  ***/
=                                                       /**********************/
=                                   /* IRB Function codes                  07/10/81 */
10  1 =   DECLARE                                    /* Constants for type code */
=           irb$sendsyn         LITERALLY '0',  /* IP->TP: send syn ctl seg */
=           irb$sendsynack      LITERALLY '1',  /* RP->TP: send syn,ack ctl segment */
=           irb$sendfin         LITERALLY '2',  /* IP->TP: send a fin ctl segment */
=           irb$sendrst         LITERALLY '3',  /* RP,IP->TP: send rst segment */
=           irb$sendcheck       LITERALLY '4',  /* RP,IP,TP->TP: try to send data */
=           irb$sendflag        LITERALLY '5',  /* RP->TP: send ctl (+data is ok) */
=           irb$timewait$to     LITERALLY '6',  /* RP->TP: delete the cdb when timer expires */
=           irb$ayt$timer       LITERALLY '7',  /* RP,IP->TP send Are-You-There signal */
=           irb$max$code        LITERALLY '7',  /* highest code: change this if any new codes */

=           irb$timeout$mask    LITERALLY '80H',/* bit is on for alarm cb's on mbx */
=           irb$ctlacb$mask     LITERALLY '40H',/* bit is on for ctl alarm acb's, not data */
=           irb$ctl$timeout$mask LITERALLY 'OCOH',/* control alarm mask combined */
=           irb$invalid$mask    LITERALLY '38H',/* mask of type buts that are not valid */
=           irb$type$mask       LITERALLY 'OFH',/* mask to check value of irb type */
=           irb$null            LITERALLY 'OFFH';/* null code for synsent handler */
    $ENDIF
11  1   DECLARE
        rbs$p       POINTER,
        rbv$p       POINTER,
        rbs BASED rbs$p
    $IF f7
    $ELSE
```

```
        $INCLUDE (:F1:TCLRBS.INC)
=           STRUCTURE(  /* Request Block for TCL Standard requests          05/29/81 */
=               contents          BYTE,    /*  0       flag: sendable data/signals here */
=               credit            BYTE,    /*  1       receive buf credit for this rb */
=               last$seq          WORD,    /*  2       seq of last seg in RB */
=                                          /* above 4 bytes hold KAOS ptr when RB on mbx) */
=               mip$buf$base      POINTER,/*  4 */
=               mip$length        WORD,    /*  8 */
=               mip$ids$id        BYTE,    /*  A = 10T */
=               mip$owner$dev$id  BYTE,    /*  B = 11T */
=               internal$process$id WORD,/* C = 12T for failure handler, not SCL process ID */
=               req               BYTE,    /*  E = 14T  Code for type of request */
=               resp              BYTE,    /*  F = 15T  reponse code: ok or error type */
=               rtn$mip$skt       WORD,    /* 10 = 16T  return address: CMX mbx or MIP socket*/
=               link              POINTER,/* 12 = 18T  optional chain to another RB*/
=               CID               WORD,    /* 16 = 22t  returned by open processing */
=               first$seq         WORD,    /* 18 = 24t  reserved for TCL: seq of 1st seg in RB  */
=               client$use        WORD,    /* 1A = 26t  Reserved for client Use (SCL) */
=               buf$len           WORD,    /* 1C = 28t  total no of client data bytes */
=               num$blks          BYTE,    /* 1E = 30t  number of data blocks */
=               vb                BYTE)    /* 1F = 31t  start of Variable-length Buffer (this byte */
=                                          /*           used only as symbolic ref for variable ptr ) */
=                                          /* 20 = 32t Total Length */
        $ENDIF
            ,
            rbv BASED rbv$p (1)
        $IF f7
        $ELSE
        $SAVE NOLIST INCLUDE (:F1:TCLRBV.INC)
            ;
        $IF f7
        $ELSE
        $INCLUDE (:F1:TCLRBC.INC)
=                                                       /**********************/
=                                                       /****  RB  Codes  ****/
=                                                       /**********************/
=               /* NOTE: These codes are still subject to change! Use these values
=               symbolically for minimum changes.  */

12  1   =   DECLARE              /* Request type codes in request blocks       09/26/81  */
=           opena$req               LITERALLY '0',
=           openp$req               LITERALLY '1',
=           close$req               LITERALLY '2',
=           status$req              LITERALLY '3',
=           def$status$req          LITERALLY '4',
=           send$req                LITERALLY '5',
=           send$eom$req            LITERALLY '6',
=           post$rbuf$req           LITERALLY '7',
=           abort$req               LITERALLY '8',
=           req$max                 LITERALLY '8',  /* must always equal max req no.*/

=           /* req values 0-127 reserved for TCL */
=
=                           /* Response codes for RB's sent back to client */
=                            /* All OK resp's are odd: equals "true" in PLM */
=           ok$resp                 LITERALLY '1',  /* No error-req accepted */
=           ok$eom$resp             LITERALLY '3',  /* No error-rcvd buff has EOM */
```

```
=       ok$fin$resp             LITERALLY '5',  /* No err - Fin rcvd (Remote close) */
=       ok$closed$resp          LITERALLY '9',  /* No error - Connection is Closed */
=                       /* Note: OK responses are designed to be OR'ed together
=                       to produce needed combinations, such as:OK/EOM/FIN = 7 */

=                       /* All Error/Abnormal event resp's are even : equals
=                       "false" in PLM */
=       invalid$req             LITERALLY '2',
=       no$resources$resp       LITERALLY '4',
=       unknown$cid$resp        LITERALLY '6',
=       buf$too$short           LITERALLY '8',  /* for status req */
=       illegal$req             LITERALLY '10', /* 0AH tried to send after close */
=       loc$abort               LITERALLY '12', /* 0CH Local client issued abort */
=       rem$abort               LITERALLY '14', /* 0EH Remote client issued abort */
=       loc$timeout             LITERALLY '16', /* 10H Local Abort timeout */
=       open$conflict           LITERALLY '18', /* 12H Tried open when matching one pending */
=   /*  close$complete          LITERALLY '18',*/   /* 12H Close sequence completed (Not an error??? */
=       invalid$pointer         LITERALLY '20'; /* 14H TCL got RB ptr or ptr in RB that is in TCL */
=   /*  already$closed          LITERALLY '22'*/    /* 16H tried Post rbuf when state=closed */
=       /* resp values 0 - 127 reserved for TCL (unless same meaning applies) */
=
    $ENDIF
    $IF f7
    $ELSE
    $INCLUDE (:F1:TCLRSC.INC)
=                                       /**********************************/
=                                       /***  RST Segment Reason Codes ***/
=                                       /**********************************/
=                                                   /* 11/23/81 */
13  1   =   DECLARE
        =       rst$old$dupl            LITERALLY   '1',    /* apparent old duplicate seg rcvd */
        =       rst$conn$closed         LITERALLY   '2',    /* CDB in closed state already */
        =       rst$no$match            LITERALLY   '3',    /* non-SYN seg doesn't match any cid */
        =       rst$zero$dest$cid       LITERALLY   '4',    /* rcvd seg had illegal dest cid */
        =       rst$syn$refused         LITERALLY   '5',    /* rvcd syn didn't match any open */
        =       rst$client$abort        LITERALLY   '6',    /* local client aborted */
        =       rst$illegal$ack         LITERALLY   '7',    /* ack not = 1 in synreceived */
        =       rst$version$mismatch LITERALLY  '8';    /* Sender's TCL version not compatible */
        $ENDIF

        /* External Procedure declarations */

14  1       setup$cdb: PROCEDURE(index, cdb$p$o) EXTERNAL;          /* in TSTART */
15  2           DECLARE   index BYTE,   cdb$p$o WORD;
16  2           END setup$cdb;

17  1       delete$cdb: PROCEDURE(cdb$index, cdb$p, rtn$code) EXTERNAL; /* In IP */
18  2           DECLARE (cdb$index, rtn$code)   BYTE,
                     cdb$p   POINTER;
19  2           END delete$cdb;

20  1       clear$lists: PROCEDURE( cdb$p,rtn$code) BYTE EXTERNAL;  /* in IP */
21  2           DECLARE rtn$code    BYTE,
                     cdb$p   POINTER;
22  2           END clear$lists;

23  1       chk$sum$calc: PROCEDURE(seg$o) WORD EXTERNAL;           /* in TCOM */
```

```
24   2          DECLARE     seg$o    WORD;
25   2          END chk$sum$calc;

26   1      check$ayt$timer: PROCEDURE(cdb$p) EXTERNAL;              /* in TP */
27   2          DECLARE cdb$p   POINTER;
28   2          END check$ayt$timer;

29   1      get$status$info: PROCEDURE(cdb$p, rbs$p) BYTE EXTERNAL; /* in IP */
30   2          DECLARE (cdb$p, rbs$p)  POINTER;
31   2          END get$status$info;

32   1      search_lcid$vector: PROCEDURE(target) WORD EXTERNAL;     /* in TCOM */
33   2          DECLARE target  WORD;
34   2          END search_lcid$vector;

35   1      gt$mod64k: PROCEDURE(n,m) BYTE EXTERNAL;                 /* in TCOM */
36   2          DECLARE (n,m) WORD;
37   2          END gt$mod64k;

38   1      ge$mod64k: PROCEDURE(n,m) BYTE EXTERNAL;                 /* in TCOM */
39   2          DECLARE (n,m) WORD;
40   2          END ge$mod64k;

41   1      max$mod64k: PROCEDURE(n,m) WORD EXTERNAL;                /* in TCOM */
42   2          DECLARE (n,m) WORD;
43   2          END max$mod64k;

44   1      min: PROCEDURE(n,m) WORD EXTERNAL;                       /* in TCOM */
45   2          DECLARE (n,m) WORD;
46   2          END min;

47   1      dlsource_eq_host: PROCEDURE(host$p) BYTE EXTERNAL;       /* in TCOM */
                                          /* function to test the source host id
                                          field in received seg against supplied
                                          host ID. returns true(matched)/false. */
48   2          DECLARE host$p  POINTER;
49   2          END dlsource_eq_host;

                                                                     /* in IP */
50   1      send$deferred$irbs: PROCEDURE(irb$index$o, irb$list$o, cdb$index, cid) EXTERNAL;
51   2          DECLARE (irb$index$o, irb$list$o, cid) WORD,
                        cdb$index   BYTE;
52   2      END send$deferred$irbs;

                                                                     /* in IP */
53   1      defer$irb$tp: PROCEDURE(type, cdb$p, irb$index$o, irb$list$o) EXTERNAL;
54   2          DECLARE type    BYTE,
                        cdb$p   POINTER,
                        (irb$index$o,   irb$list$o) WORD;
55   2          END defer$irb$tp;

56   1      stky_incr: PROCEDURE(wd$p) EXTERNAL;                     /* in tcom */
57   2          DECLARE wd$p    POINTER;
58   2          END stky_incr;

            $IF log
            $ENDIF
```

```
$IF f7
$ELSE
$SAVE NOLIST INCLUDE (:F1:KAOS.DCP)
$IF f7
$ELSE
$SAVE NOLIST INCLUDE (:F1:DLL.DCP)
$IF f7
$ELSE
$SAVE NOLIST INCLUDE (:F1:MIP.DCP)
$IF f7
$ELSE
$SAVE NOLIST INCLUDE (:F1:THACF.INC)

$IF log
$ENDIF

$IF log
$ENDIF

160  1    DECLARE
             rp$irb$list (3) BYTE    INITIAL(0FFH,0FFH,0FFH),
             rp$irb$index    BYTE    INITIAL(0);
                                                  /*************************/
                                                  /*** rp$defer$irb$type ***/
                                                  /*************************/
161  1    rp$defer$irb$tp: PROCEDURE(type);
162  2    DECLARE type BYTE;                  /* irb type to send */

163  2    CALL defer$irb$tp(type, cur$cdb$p, .rp$irb$index, .rp$irb$list);
164  2    END rp$defer$irb$tp;


                                                  /*************************/
                                                  /**   clear$cdb$alarms  **/
                                                  /*************************/

165  1    clear$cdb$alarms: PROCEDURE (cca$cdb$p) PUBLIC;
                                             /* shared code to kill both alarms in
                                             the specified connection data base */
166  2    DECLARE
             cca$cdb$p   POINTER,
             cca$c   BASED cca$cdb$p
         $IF f7
         $ELSE
         $SAVE NOLIST INCLUDE (:F1:TCLCDB.INC)
             ;
167  2    CALL cq$clear$alarm(@cca$c.data$alarm$cb);          /* clear the re-tran timer */
168  2    CALL cq$clear$alarm(@cca$c.ctl$alarm$cb);           /* kill the control timer */
169  2    if (cca$c.data$acb$flag=0) or (cca$c.ctl$acb$flag=0) then
170  2      call Cqhaltandcatchfire(9004h);

171  2    END clear$cdb$alarms;
```

```
                                                    /*************************/
                                                    /** try$to$delete$cdb  ***/
                                                    /*************************/

172   1     try$to$delete$cdb: PROCEDURE(ttd$cdb$index, ttd$cdb$p, resp$code) PUBLIC;
                                          /* common code for deleting a connection
                                          (if there is an RB to return) or putting
                                          it into the Closed state (to wait for
                                          an RB) if not. */
173   2     DECLARE
                ttd$cdb$index   BYTE,
                ttd$cdb$p   POINTER,
                resp$code   BYTE,
                ttd$c BASED ttd$cdb$p
            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:TCLCDB.INC)
                ;
174   2     ttd$c.state = closed;               /* mark it closed in case delete fails */
175   2     spec$type(ttd$cdb$index) = 0FFH;    /* Remove CDB from match list */
176   2     ttd$c.closed$reason = resp$code;    /* Save the reason why its deleted/closed */
177   2     CALL clear$cdb$alarms(ttd$cdb$p);   /* kill both its alarms */

                        /* NOTE: need to re-do this and delete$cdb to eliminate overlap */

178   2     IF clear$lists(ttd$cdb$p, resp$code) THEN       /* send back all client's RB's */
179   2         CALL delete$cdb(ttd$cdb$index, ttd$cdb$p, resp$code );
180   2     END try$to$delete$cdb;

                                                    /*********************/
                                                    /**   accept$conn   */
                                                    /*********************/

181   1     accept$conn: PROCEDURE(state$code, tp$req$code);
182   2     DECLARE
                (state$code, tp$req$code)   BYTE;
                                          /* Fill in cdb fields from received */
                                          /* segment in listen or synsent */
                                          /* kill timers unless we are entering
                                          syn received state from syn sent, AND the
                                          syn we just received doesn't ack our
                                          previous syn.  This attempts to take care
                                          of the case where our earlier SYN was
                                          rejected due to no matching cdb, but then
                                          he did an active open while we are doing
                                          a control timeout to re-send our syn. */
183   2         IF (state$code <> synrcvd) OR (rp_.seg$ack$no = 1) THEN
184   2             CALL clear$cdb$alarms(cur$cdb$p);     /* kill any active timers */
185   2         c.rem$net = loc$net;            /* Net ID not available from DL! */

186   2         CALL MOVW(@rp_.dl$source0, @c.rem$host(0), 3);  /* copy host ID into CDB */

187   2         c.rem$port = rp_.source$port;
188   2         c.rem$cid = rp_.source$cid;
189   2         c.his$credit = rp_.ctl AND credit$mask;
190   2         c.my$ack$no = rp_.seg$seq$no;
```

```
191   2         c.state = state$code;           /* Put Connection Into New State */
                $IF log
                $ENDIF
192   2         c.no$confid = 0;                /* reset the retries counter */
193   2         c.cum$retran$dw = 0;            /* clear the cummulative retry time */
194   2         spec$type(cur$cdb$index) = spec$type(cur$cdb$index) OR 80H; /* remove from */
                                              /* conn match list in a way that allows */
                                              /* it to be added back if RST in syn-rcvd */
                                              /* state encounterd and we came from listen */
                                              /* Now get an irb, tell TP what to send */
                                              /* unless its null (from synsent) */
195   2         IF tp$req$code <> irb$null THEN CALL rp$defer$irb$tp(tp$req$code);
197   2      END accept$conn;

                                                        /*********************/
                                                        /** send$rst$reply  **/
                                                        /*********************/
198   1      send$rst$reply: PROCEDURE(reason$code);
                                    /* Tell Transmit Process to send an RST segment */
                                    /* in reply to a rcvd segment. Always checks */
                                    /* that there is not an RST in current rcvd seg, */
                                    /* since it is never permissible to reply to an */
                                    /* RST seg with an RST (causes infinite packet */
                                    /* exchange between the two nodes). */
199   2      DECLARE reason$code BYTE;

200   2         IF (rp_.ctl AND rst$mask) <> 0 THEN
201   2            DO;
                $IF log
                $ENDIF
202   3               RETURN;
203   3            END;
204   2         CALL cq$signal(.sched$lock);        /* relinquish lock while we get buf */
205   2         lirb$p = cq$receive(.free$lirb$mbx);
206   2         CALL cq$waitsem(.sched$lock);       /* now re-acquire the lock */
207   2         lirb.type = irb$sendrst;
208   2         lirb.reason = reason$code;
                                          /* Copy the fields we need to send an
                                          RST reply into lirb.  Fields in lirb
                                          are in same sequence and length as
                                          the needed seg header fields.  Subtract
                                          5 from length for ptr and type. */
209   2         lirb.dl$dest(0)   = rp_.dl$source0;
210   2         lirb.dl$dest(1)   = rp_.dl$source1;
211   2         lirb.dl$dest(2)   = rp_.dl$source2;
212   2         lirb.dest$port    = rp_.source$port;
213   2         lirb.source$port  = rp_.dest$port;
214   2         lirb.dest$cid     = rp_.source$cid;
215   2         lirb.source$cid   = rp_.dest$cid;
216   2         lirb.seg$seq$no   = rp_.seg$ack$no + 1;  /* make it acceptable, not a duplicate */
217   2         lirb.seg$ack$no   = rp_.seg$seq$no;

218   2         CALL cq$send(.tp$mbx, lirb$p);
219   2      END send$rst$reply;

                                                  /************************/
```

```
                                                        /** estab$rst$chk$false */
                                                        /************************/
220   1     estab$rst$chk$false: PROCEDURE BYTE;
                                        /* Function to check the received packet
                                        in a synchronized state (except SYN Rcvd)
                                        to close the connection if it contains a
                                        Reset flag */
221   2     IF (rp_.ctl AND rst$mask) <> 0 THEN
222   2         DO;                           /* set state, clr lists, clr alarms */
223   3             CALL try$to$delete$cdb(cur$cdb$index, cur$cdb$p, rem$abort);
            $IF log
            $ENDIF
224   3             RETURN (false);
225   3         END;
226   2     ELSE RETURN (true);
227   2     END estab$rst$chk$false;


                                                      /*********************/
                                                      /**      put        */
                                                      /*********************/
228   1     put: PROCEDURE;

                  /* This routine does full-up re-assembly of received segments
                  into client's receive buffers; multiple segments will be stored
                  into any number of buffers (RB'S) and blocks within those buffers
                  contiguously, without regard for segment, block, or buffer
                  boundaries.  In addition, if there is insufficient receive buffer
                  space to hold a segment, partial segments are accepted and saved,
                  with the remainder of the segment being stored when additional
                  receive buffer space is available.  */

                  /* Note: this re-assembly policy acks received segents when they
                  can be stored into client data buffers, rather than when the data
                  is actually passed to the client's return mailbox.  This may have
                  significance at Abort times, if the remote client does an abort
                  after having received an ACK of his Send data.  The data may not
                  have actually been sent to the local client yet, although he will
                  get it when the abort is processed.  Note that if the remote client
                  sent an EOM along with the last data message, then the local TCL
                  WILL put it on the mailbox when it finishes processing the segment.
                  */

229   2     DECLARE
              ctl$bits$present    BYTE,
              cb$data$p           POINTER,          /* current block data ptr */
              cb$data BASED cb$data$p (1) BYTE,   /* The client's rcv buffer array */
              copy$len            WORD,
              rp$len$left         WORD,
              put$rbs$p           POINTER,
              put$rbv$p           POINTER,
              put$rbs BASED put$rbs$p
            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:TCLRBS.INC)
              ,put$rbv BASED put$rbv$p (1)
            $IF f7
```

```
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:TCLRBV.INC)
                ;

                                        /* ROUTINES FOR PUT */
                                                                /**  setup$new$blk  **/
230   2     setup$new$blk: PROCEDURE;
                           /* sets up the variables associated with a new block
                           in a Request Block buffer. */
231   3     IF c.curblk$index < put$rbs.num$blks THEN
232   3         DO;                                   /* there's really a blk there */
233   4             c.cb$data$index = 0;              /* start storing in first byte */
234   4             c.curblk$len$left = put$rbv(c.curblk$index).blk$len;
            $IF mipform
235   4             cb$data$p = cq$mip$get$address( put$rbv(c.curblk$index).blk$ptr);
            $ELSE
            $ENDIF
236   4         END;
237   3     ELSE
              c.curblk$len$left = 0;                  /* no blk so no blk len */

238   3     END setup$new$blk;


                                                        /**********************/
                                                        /**  send$rbs$back  **/
239   2     send$rbs$back: PROCEDURE(insert$ctl$bits$flag) BYTE;
                           /* this routine is contained in PUT; it takes the top RB off
                           the pcbq and sends it back to the client.
                           If the received segment contains sequence-controlled control
                           signals (EOM and FIN, currently) and the caller has indicated
                           that such signals should be passed to the client, the RB
                           will be marked appropriately.  A new RB is set up for receiving,
                           if one is available. A "true" is returned if there was another
                           RB, otherwise a "false" is returned. */

240   3     DECLARE insert$ctl$bits$flag    BYTE;

241   3     put$rbs.resp = ok$resp;
242   3     IF insert$ctl$bits$flag THEN
243   3         DO;
244   4             IF (rp_.ctl AND eom$mask) <> 0 THEN put$rbs.resp = ok$eom$resp; /* tell client there's an EOM */
246   4             IF rp_has_fin THEN put$rbs.resp = put$rbs.resp OR ok$fin$resp;
248   4         END;
249   3         c.pcbq$hdr = put$rbs.link;      /* take top rb off the queue, insert 2nd one */
250   3         put$rbs.link = 0;               /* always zero the link field */
251   3         c.my$credit = c.my$credit - put$rbs.credit;     /* reduce my rcv buf credit */
                                                /* by the amount of segs in this RB */
252   3         CALL cq$send(.buf$mip$mbx, put$rbs$p);
253   3         c.pcbq$buf$cnt = c.pcbq$buf$cnt - 1;

254   3         IF c.pcbq$buf$cnt <> 0 THEN     /* there's another RB - set it up */
255   3             DO;
256   4                 c.curblk$index = 0;     /* start with first block in new buffer */
257   4                 put$rbs$p = c.pcbq$hdr;
258   4                 put$rbv$p = @put$rbs.vb;
259   4                 CALL setup$new$blk;     /* initialize the data ptrs, cnts */
```

```
260   4                 RETURN(true);
261   4             END;
262   3     RETURN(false);                      /* returns false if no RB avail */

263   3     END send$rbs$back;


264   2     get$next$blk: PROCEDURE(pkt$empty) BYTE;
                                  /* this routine tries to set up a new buffer
                                  block to copy into.  If there are no more blocks
                                  in the current RB, it calls send$rbs$back to get
                                  the next RB, if any. If there was a new block or
                                  RB, this routine returns "true"; if not, it
                                  returns "false" */
265   3     DECLARE
                pkt$empty   BYTE;       /* flag: true if all data from pkt has been stored */

266   3     c.curblk$index = c.curblk$index + 1;    /* bump the blk index to see if there's another */
267   3     IF c.curblk$index >= put$rbs.num$blks THEN      /* if true, then no more blks in */
                                                        /* this rb, so try to get a */
                                                        /* new RB */
                                                /* A param of "false" => no EOM or FIN */
                                                /* in pkt; a return of fasle => no more RBs */
268   3         RETURN (send$rbs$back(pkt$empty AND ctl$bits$present) );
269   3     ELSE
                DO;                             /* there's another blk in this RB: set it up to */
270   4             CALL setup$new$blk;
271   4             RETURN(true);
272   4         END;

273   3     END get$next$blk;


                        /*  ****    Main-line code for PUT    **** */

                                                /* first, check that segment seq no */
                                                /* is expected value: without re-assembly */
                                                /* buffers, seg's can only be stored in */
                                                /* strict sequence. */
274   2     IF rp_.seg$seq$no <> (c.my$ack$no+1) THEN
275   2         DO;
          $IF log
          $ENDIF
276   3             RETURN;
277   3         END;
278   2     IF c.pcbq$buf$cnt = 0 THEN              /* no buffers at all - don't bother */
279   2         DO;                               /* unless its a FIN without data or other */
                                                /* sequence-consuming controls-only EOM, now */
280   3             IF (rp_.seg$data$len <> 0) OR ((rp_.ctl AND eom$mask) <> 0) THEN
281   3                 DO;                       /* bump the "no buf space" cntr */

282   4                     CALL stky_incr(@c.rcv$buf$rej$cnt);
283   4                     CALL stky_incr(@tot$rcv$buf$rej);
                                                /* also record no of bytes outstanding */
284   4                     IF c.rcv$bytes$consumed = 0 THEN
285   4                         c.pending$rcv$data = rp_.seg$data$len;
          $IF log
```

```
            $ENDIF
286   4                 RETURN;
287   4             END;
288   3           IF rp_has_fin THEN              /* its a lone FIN: we can ack that, since */
289   3             DO;                           /* it is "received" into the conn state */
290   4               c.my$ack$no = rp_.seg$seq$no;   /* mark it acked for caller */
291   4               CALL rp$defer$irb$tp(irb$send$flag);    /* tell tp to ACK it */
292   4               RETURN;
293   4             END;
294   3           RETURN;                         /* safety net - we shouldn't get here */
295   3         END;
296   2       c.pending$rcv$data = 0;             /* clear the "number of bytes undelivered cnt" */
                                                  /* Set up "from" data variables */
297   2       rp$len$left = rp_.seg$data$len - c.rcv$bytes$consumed; /* # of bytes to be copied from */
                                                  /* seg - we may have stored some the last */
298   2       ctl$bits$present = (rp_.ctl AND eom$fin$mask) <> 0; /* note FIN implies EOM */
                                                  /* time the segment was sent to us (don't ack until its all stored
              -) */
                                                  /* set up "to" data variables */
299   2       put$rbs$p = c.pcbq$hdr;             /* fixed part of ptr to RB */
300   2       put$rbv$p = @put$rbs.vb;                /* variable part */
            $IF mipform
301   2       cb$data$p = cq$mip$get$address(  put$rbv(c.curblk$index).blk$ptr ); /* Starting store address */
            $ELSE
            $ENDIF


302   2       DO forever;                       /* now, go into the main loop trying to */
                                                /* copy receive data into client buffer blocks */
303   3         copy$len = min(rp$len$left, c.curblk$len$left); /* bytes of data we can copy this pass */
304   3         IF copy$len <> 0 THEN
305   3           DO;
                                                  /* Do the primary data move here */
                  /* NOTE: NEED TO SET HARDWARE ARBITRATOR FLAG FOR HIGH SPEED ON COMM BD */
306   4             CALL MOVW(@rp.seg$data(c.rcv$bytes$consumed),
                                                    @cb$data(c.cb$data$index), copy$len/2);
                  /* NOTE: THEN NEED TO CLEAR THE HARDWARE FLAG */
                                                  /* see if we have an odd byte to move */
307   4               IF copy$len THEN            /* if true, then lsb of copylen is set, */
                                                  /* so there was an odd byte */
308   4                 cb$data(c.cb$data$index + copy$len - 1) =
                              rp.seg$data(c.rcv$bytes$consumed + copy$len - 1);

                                                  /* now update all the offsets and counts */
309   4               c.cb$data$index = c.cb$data$index + copy$len;   /* the "to" offset */
310   4               c.rcv$bytes$consumed = c.rcv$bytes$consumed + copy$len; /* the "from" count */
311   4               put$rbs.buf$len = put$rbs.buf$len + copy$len;   /* total bytes in this RB */
312   4               rp$len$left = rp$len$left - copy$len;   /* bytes remaining in rcvd seg. */
313   4               c.curblk$len$left = c.curblk$len$left - copy$len; /* bytes remaining on rcv blk */
314   4             END;
                                                  /* At this point, either rp$len$left or */
                                                  /* c.curblk$len$left MUST be zero. */

                                                  /* now see if we're thru with this pkt */
315   3         IF rp$len$left = 0 THEN                 /* yes - marked the seq no ack-able */
316   3           DO;
317   4             c.rcv$bytes$consumed = 0;       /* zero to start next seg */
```

```
318   4                 c.my$ack$no = rp_.seq$seq$no;   /* mark the seq no to be acked */
319   4                 CALL rp$defer$irb$tp(irb$send$flag);    /* tell tp to ACK it */
                                        /* NOTE: consider delaying this ACK */
                                        /* later if it helps: maybe if rcvd pkt */
                                        /* doesn't have EOM, and my rcv window <> 0, */
                                        /* and round-trip delay seems to be high. */

                                        /* now see if we need to set up more buf */
                                        /* space for next pkt... will send RB back */
                                        /* if no more space in this RB, or if */
                                        /* pkt had controls which force it to be */
                                        /* returned. */
320   4                 IF c.cur$blk$len$left = 0 THEN
321   4                     scratch = get$next$blk(true);   /* param=true => pkt is empty -  */
                                                            /* ignore result since pkt empty */
322   4                 ELSE IF ctl$bits$present THEN
323   4                     scratch = send$rbs$back(true);  /* EOM and empty pkt forces buffer to be returned */
324   4                 RETURN;
325   4             END;
                                        /* If we get here, then c.curblk$len$left MUST be zero: */
326   3         ELSE                      /* pkt not empty : want to store more data */
                      DO;
327   4                 IF NOT get$next$blk(false) THEN
328   4                     DO;           /* there isn't any more rcv buffer space, so */
                                        /* note amount of bytes left in pkt and quit */
329   5                     c.pending$rcv$data = rp$len$left;
          $IF log
          $ENDIF
330   5                     RETURN; /* wait til retransmission and hope there's more buffer space */
331   5                     END;
332   4             END;
333   3     END;    /* of forever loop */

334   2     END put;


                                                        /*********************/
                                                        /**     complete    */
                                                        /*********************/
335   1     complete: PROCEDURE(target$seg$no);
                  /* This routine tries to return ACK'd transmit buffers to the
                  local client.  It accepts a sequence number, and attempts
                  to remove, and return to the client, all cbtq RB's with data
                  corresponding to the same or lower (modulo 64K) sequence numbers.
                  All seq's in an RB must be less than or equal to the offered one
                  for the RB to be returned.  Exception: Close RB's are not returned
                  when Complete acks them, since they are held until the CDB is
                  deleted to simplify the client interface.  The transmit RB buf count
                  is, however, zeroed at ack of Close to prevent the Transmit
                  Process from sending it again. */


336   2     DECLARE
                top$rbs$p       POINTER,
                target$seg$no   WORD,
                top$rbs         BASED top$rbs$p
          $IF f7
```

```
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:TCLRBS.INC)
                ;

337   2     IF c.cbtq$buf$cnt = 0 THEN RETURN;  /* nothing to remove */
339   2     top$rbs$p = c.cbtq$hdr;

340   2     DO WHILE ge$mod64k(target$seg$no, top$rbs.last$seq);

                                    /* send back the top RBS */
341   3         top$rbs.resp = ok$resp;
342   3         IF top$rbs.req = close$req THEN /* Its a Close RB - don't send it back */
343   3             DO;
344   4                 c.cbtq$buf$cnt = 0;     /* force xmit RB count to zero */
345   4                 RETURN;
346   4             END;
347   3         c.cbtq$hdr = top$rbs.link;
348   3         top$rbs.link = 0;               /* always zero the link field */
349   3         CALL cq$send(.buf$mip$mbx, top$rbs$p);
350   3         IF (c.cbtq$buf$cnt := c.cbtq$buf$cnt - 1) = 0 THEN RETURN;  /* queue is empty */
352   3         top$rbs$p = c.cbtq$hdr;
353   3     END;
354   2     END complete;

            $EJECT
```

```
                                                        /**********************/
                                                        /**  get$round$trip  */
                                                        /**********************/
355   1     get$round$trip: PROCEDURE;

                                        /* code-saver routine to compute the
                                        roundtrip time of the ack of our segment,
                                        convert to setalarm time units, and clr
                                        the timed seq no */

                                        /* set$retran flag added to allow
                                        the initial open sequence roundtrip
                                        time to be computed even when it took
                                        just one retransmission */

                                        /* The above flag is removed */

356   2     CALL cq$read$clock(.rp$timestamp);
357   2     c.timed$seq$no = 0;                       /* clear it so TP knows it can be used*/
358   2     rp$roundtrip = rp$timestamp - c.seg$trans$time$dw;
                                        /* if we had clock wrap-around, then
                                        high-order bit of rp$roundtrip will be
                                        set (can test upper byte to see): if so
                                        then fix it by complementing roundtrip. */
359   2     IF HIGH(HIGH(rp$roundtrip)) >= 80H THEN rp$roundtrip = - rp$roundtrip;
                                        /* multiply by 32 to get it to setalarm
                                        timeunits, and multiply by 2 to make
                                        the retransmit time average out to twice
                                        the roundtrip time to avoid extra sends */
361   2     rp$roundtrip = SHL(rp$roundtrip,6);     /* works out to a shift left 6 */

                        /* In order to adapt to each of two cases (1-rcv buf is
                        consistently posted just after the segment is sent, necessitation
                        one retransmission, 2-long-haul networks where we occasionally
                        get have a timeout value which is too low, resulting in
                        saturation of the network until we re-adapt higher), but
                        avoid the retransmission timer growing exponentially due
                        to having one re-transmission of each data seg, we want
                        to take special action if there was exactly one timeout before
                        the segemtn was acknowledged.  In this case, we subtract the
                        previous timeout value from the computed roundtrip time to
                        get the time it took (discounting occasional packet loss) for
                        the ACK of the second segment to arrive.  This will result in
                        the average timeout going down. */

                        /* SO, if there was exactly one retransmission,
                        we subtract off the previous retranmsit
                        timer value from the computed roundtrip timeout, except if
                        the is the initial open sequence timing,  in which case
                        we need to return an accurate  value */

                        /* George's algorithm is further modified - to special case in
                        the above fashion for two retransmission - because this was
                        the case with NDS-II.  With 1000 bytes posted buffers the
                        first packet was lost because the buffer was not posted in
                        time and the 2nd retransmission is required because the first
                        retransmission is not completely consumed and thus not acked
```

```
                        Rajesh Shah */

                        /* 12/14 A completeley new algorithm is implemented - and
                        is described in the procedure Update$retransmit$timeout */

362   2       END get$round$trip;

                                                        /*************************/
                                                        /**  set$retran$timeout **/
                                                        /*************************/
363   1       set$retran$timeout: PROCEDURE;
                                        /* routine to compute the round-trip
                                        segment time from the connection handshake,
                                        and set the connection's retransmit timeout
                                        value accordingly */
364   2       CALL get$round$trip;             /* compute the round trip time of ack */
                                               /* now force retran time to the weighted */
365   2       c.retran$to$dw = rp$roundtrip;   /* roundtrip time just computed */
366   2       END set$retran$timeout;

                                                        /*************************/
                                                        /* update$retran$timeout */
                                                        /*************************/
367   1       update$retran$timeout:  PROCEDURE;
```

```
/*****************************************************************
                TCL  Adaptive  Retransmission   Algorithm

    Rajesh Shah                                          Dec 12th 1981

    TCL's retransmission timeout must be adapted to  the  conditions
    posed by the receiver and the line.

    TCL  will  need  to  retransmit  for  one of the following three
    reasons:


        1.  Occassional packet loss.

        2.  The receiver does not post a receive buffer in time,
            so that when  the  packet arrives there are  no  receive
            buffers available to TCL.

        3.   The  total number of bytes in the remaining receive
            buffers is less    than  the number of bytes in the packet
            being transmitted.

        The  Retransmission  Adaptation  Algorithm  must  adapt  the
    retransmission timeout to all the above three cases.

    For Occassional packet loss, the ideal retransmission timeout is
    little  more than the roundtrip time of a packet.  (Time the ack
```

of the packet arrives - time at which the packet was sent).

If the packet is dropped because it arrived earlier than a receive buffer was posted, the ideal retransmission time is the average time it takes for the receiver to post buffers.

If the packet needs to be retransmitted because the receive buffer did not have sufficient bytes to completely consume a packet, the ideal retransmission time is again the time it would take the receiver to post the next buffer.

The last situation is created both by posting buffers smaller than packet sizes (when full packets are expected) or by posting buffers larger than packet sizes in which cases the first few packets will be consumed immediately requiring no retransmission, and the next packets will have to be retransmitted if the buffer size is not a multiple of packet size. In the first case, each posted buffer will require n retransmissions where n is 1 + packet size / buffer size (some will require n-1 depending on boundary conditions). The second case requires m packets to be transmitted without retransmissions (not counting packet loss) and the m+1 th packet to be retansmitted once.

The roundtrip time can be easily computed by time each segment and its ack, however it is necessary to make an estimate of the average time it takes the receiver between posting buffers, if retransmission are necessary.

The goal here is to approximately reach the range and let the retransmission timeout oscillate in close viscinity. We attempt to reach closer to this timeout by the following procedure.

Let us say that We detect that we are transmitting much more often than required (we will discuss how to detect this). If this is the case, then increasing the retransmit timeout should reduce the number of retransmissions. We double the retransmission timeout and see if the number of retransmissions reduce. If we do than we are on the right track. We continue to double retransmission timeout until, the number of retransmissions stop reducing. This implies that the retransmission timeout is larger than the time the receiver needs to post a buffer. Now when the number of retransmissions is constant we be optimistic and reduce the timeout by 12.5. It is easy to detect that we are transmitting too often. Since we are reducing by 12.5 when the number of retransmissions are constant, at some boundary the number of retransmission will double. Also if the receiver starts posting buffer more slowly now, the number of retransmissions will suddenly go up. We detect this by comparing out retransmissions with the previous number of retransmissions and if it has increased we go into a increase$timeout$state and stop when we find the number of retransmissions do not decrease with increasing timeout.

When there are no retransmission we gradually take the timeout close to twice the roundtrip time or do nothing at all to save

```
                    code and to save ourselves from dword integer arithmetic.

                    The above discussion assumed that full size  packets  are  being
                    sent.  However the same arguments apply for any size packets.

                ****************************************************************/

368   2         Increase$ret: PROCEDURE;
                    /* Code saver common procedure to Increase the retransmission timeout
                    Increase by 12.5% to 50% */

369   3             c.retran$to$dw = c.retran$to$dw + SHR (c.retran$to$dw, Retran$increase);
370   3         end Increase$ret;

371   2         Decrease$ret: PROCEDURE;

          /*        declare DW dword,
                            (DW$lo$w, DW$hi$w) word AT (@DW),
                            (DW$lo$byte, DW$hi$byte) byte AT (@DW);
          */
                    /* Common prodecure to decrease retransmission time by 12.5 % */

372   3             c.retran$to$dw = c.retran$to$dw - SHR (c.retran$to$dw, 3);

                    /* If the resulting timeout value is too small then - the algoritm
                    runs the risk of failing - because at values where TCL's compute
                    time becomes significant, reducing timeout does not increase
                    number  of retransmissions.  Hence a lower bound must be implemented.
                    This value has been found by experience to be around 10 thousand
                    800 ns clicks for Ethernet, more for DBP.
                    If the timeout is found lower than this we make
                    it twice the minimum and change the state to Timeout$increase state */

          /*        DW = c.retran$to$dw;

                    IF DW$hi$w = 0  AND  DW$hi$byte < 40  THEN DO;
                          ** Note that 40 x 256 = 10k **
                        DW$hi$byte = 80;
                        c.retran$to$dw = DW;
                        c.retransmit$state = Timeout$increase$state;
                    END;

          */
373   3             IF c.retran$to$dw < min$retran$time  THEN DO;
375   4                 c.retran$to$dw = SHL(min$retran$time,1);
376   4                 c.retransmit$state = Timeout$increase$state;
377   4             END;
378   3         end Decrease$ret;


                                /* checks to see if the ack received applies
                                to the segment being timed (if any).  If so,
                                updates this connection's retran timer.*/

                                /* Also if there are no retransmissions then do not
                                change the retransmission timeout. */
```

```
379   2       IF c.timed$seq$no = 0
380   2               or c.no$confid = 0 THEN RETURN;

                    /* Note that the retansmit$state is not modified and the the
                    last$no$confid is not replaced by 0.  This is done so that we compare
                    current num of retransmissions with the last time we had non zero
                    retransmissions */

381   2       IF c.retransmit$state = Timeout$Increase$state THEN DO;

383   3           IF (c.no$confid + 1) < c.last$no$confid or (c.no$confid = 0ffffh)
                        /* Number of retransmissions has decreased so continue to
                        Increase the retransmission timeout.  Note that a difference of
                        1 is not considered significant  - This also introduces hysteresis
                        in the algorithm */

384   3             THEN CALL increase$ret;

385   3           ELSE DO;

                        /* Change the retransmit state to Timeout$steady$state - in this
                        state it reduces every time by 12.5 %. Also reduce it now by
                        12.5 % twice because increasing by 50% did not do any good */

386   4                 c.retransmit$state = Timeout$Steady$state;
387   4                 call decrease$ret;
388   4                 call decrease$ret;

389   4           END;

390   3         END;

391   2       ELSE DO;     /* It is steady state - change to increase state if the
                            number of retransmissions have increased by more than 1 */

392   3           IF (c.no$confid > c.last$nc$confid + 1) or c.no$confid = 0ffffh
393   3             THEN DO;

394   4                 c.retransmit$state = Timeout$increase$state;
395   4                 CALL Increase$ret;

396   4               END;

397   3           ELSE call Decrease$ret;

398   3       END;

399   2       c.last$no$confid = c.no$confid;


400   2       END Update$retran$timeout;



                                                    /**********************/
                                                    /**   rp$err$flow    **/
                                                    /**********************/
```

```
401   1     rp$err$flow: PROCEDURE;
                                        /* Established state logic - this is the
                                        receive side of the error and flow control
                                        algorithm; it is shared by several states */
402   2     DECLARE
                old$his$ack$no  WORD,
                old$his$credit  WORD;

403   2     IF rp_has_ack THEN                /* seg ack field is significant, so */
404   2         DO;                           /* process it.*/
405   3             IF ge$mod64k(rp_.seg$ack$no, c.his$ack$no) THEN     /* remote client */
                                        /* may have ACKed some data - save credit */
                                        /* info, since ack is not obsolete */
406   3                 DO;
407   4                     old$his$ack$no = c.his$ack$no;  /* note current values for */
408   4                     old$his$credit = c.his$credit;  /* comparison below */
409   4                     c.his$credit = (rp_.ctl AND credit$mask);   /* save remote credit */
                                        /* if the seg actually acks some */
                                        /* additional data, update our cdb. */
410   4                     IF gt$mod64k(rp_.seg$ack$no, c.his$ack$no) THEN
411   4                         DO;
412   5                             c.his$ack$no = rp_.seg$ack$no;
413   5                             CALL complete(c.his$ack$no);    /* take my acked xmit pkts off the xmit queue */
                                        /* now check if a seg we are timing was acked - */
414   5                             CALL update$retran$timeout; /* if so, compute new timeout */

                                        /* Since his$ack$no changed, update nxt */
                                        /* seq no to send to be one greater than */
                                        /* highest one acked */
                                        /* NOTE: compute it with segackno, since its the same
                                        value, and rp_.seg$ack$no is avail with fewer code bytes */
415   5                             c.next$transmit = max$mod64k(c.next$transmit, rp_.seg$ack$no + 1);
                                        /* Cancel data timer since we got an ack of top data */

416   5                             CALL cq$clear$alarm(@c.data$alarm$cb);
                                        /* Now re-start it on the oldest remaining */
                                        /* xmit queue segment, if there is one. */
417   5                             IF c.cbtq$buf$cnt <> 0 THEN
418   5                                 DO;
419   6                                     c.data$acb$irb$type = irb$send$check OR irb$timeout$mask;
420   6                                     CALL cq$set$alarm(@c.data$alarm$cb, .tp$mbx,
                                                  high(c.retran$to$dw),
                                                  low ( c.retran$to$dw));
421   6                                 END;
422   5                             c.no$confid = 0;    /* reset the retry count */
423   5                             c.cum$retran$dw = 0;/* clear the cummulative retry time */
424   5                         END;
                                        /* if there is still something on xmit
                                        queue and credit or ack info on remote guy
                                        has changed due to this rcvd segment,
                                        tell TP about it, in case it allows
                                        another seg to be sent */
425   4                 IF c.cbtq$buf$cnt <> 0 AND
                          ( (old$his$ack$no <> c.his$ack$no) OR
                            (old$his$credit <> c.his$credit) )
426   4                  THEN CALL rp$defer$irb$tp(irb$send$check);
427   4             END;
```

```
428   3          END;
                                        /* Now send rcvd pkt to the rcv buff */
                                        /* mgr,if there is text, EOM or FIN */
                                        /* to try to store it in client buf */
429   2       IF (rp_.seg$data$len <> 0) OR ((rp_.ctl AND eom$fin$mask) <> 0) THEN CALL put;

431   2       CALL check$ayt$timer(cur$cdb$p);    /* protect against remote guy dying */

432   2       END rp$err$flow;

                                                  /****************************/
                                                  /**  chk$deferred$status   **/
                                                  /****************************/

433   1       chk$deferred$status: PROCEDURE(cds$cdb$p) PUBLIC;
                                        /* routine to check if there are any deferred
                                        status requests posted, and return them
                                        (with status info, if requested) to host
                                        client process */
434   2       DECLARE
                  rbs$p       POINTER,
                  cds$cdb$p   POINTER,
                  rbs BASED rbs$p
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLRBS.INC)
                  ,
                  cds$c BASED cds$cdb$p
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLCDB.INC)
                  ;

435   2       rbs$p = cds$c.def$status$p;         /* set up addressing to top RBS */
436   2           DO WHILE rbs$p <> 0;            /* return all (if any) def stat rb's */
                                                  /* update list head to next RB, if any, */
437   3               cds$c.def$status$p = rbs.link;  /* or insert zero if this is the last one */
438   3               rbs.link = 0;
439   3               rbs.resp = ok$resp;
                                                  /* send back status info if client asked for it */
440   3               IF rbs.num$blks <> 0 THEN rbs.resp = get$status$info(cds$cdb$p, rbs$p);
442   3               CALL cq$send(.buf$mip$mbx, rbs$p);
443   3               rbs$p = cds$c.def$status$p; /* set up for next RB, if any */
444   3           END;

445   2       END chk$deferred$status;

                                                        /**********************/
                                                        /** process$segment **/
                                                        /**********************/

446   1       process$segment: PROCEDURE;


447   2       rest$of$finwait$2: PROCEDURE;
                                              /* routine to handle the finwait2 state */
                                              /* logic so that it can be shared with */
                                              /* finwait1 if fw1 goes to fw2. */
```

```
448   3             IF rp_has_fin THEN
449   3                 DO;
450   4                     IF rp_.seg$seq$no = c.my$ack$no THEN        /* our Fin was ACKed so enter new state */
451   4                         DO;
452   5                             c.state = timewait;
          $IF log
          $ENDIF
                                                /* now set timer for going to Closed state */
453   5                             CALL clear$cdb$alarms(cur$cdb$p);
                                                /* note we mark acb as ctl type for TP's use */
454   5                             c.ctl$acb$irb$type = irb$timewait$to OR
                                                           irb$timeout$mask OR
                                                               irb$ctlacb$mask;
                                                /* set the 2 MSL timeout, enter closed state */
                                                /* when it expires - just waiting on re-xmit */
                                                /* of FIN from remote guy so the pre-filter */
                                                /* above can ack it again for him.  Set it to */
                                                /* bigger than retran sometime */
455   5                             CALL cq$set$alarm(@c.ctl$alarm$cb, .tp$mbx,
                                                    high(c.retran$to$dw),
                                                    low (c.retran$to$dw));
456   5                         END;
457   4                 END;

458   3       END rest$of$finwait$2;

                                                                        /**  fin$seq$no **/
459   2       fin$seq$no: PROCEDURE WORD;
                                    /* Used in Finwait1 and Closing state: returns the
                                       sequence number of the FIN associated with the local
                                       close which has already occured.  Tricky because
                                       of possible states: (1) we may not yet have sent the
                                       FIN, either because there was a non-zero xmit queue,
                                       or because a FIN may have arrived between the time IP
                                       enters FW1 and when TP bumps the highest-sent value;
                                       or (2) the FIN may have been sent (the current rcvd
                                       pkt may or may not ack it), or (3) the FIN may already
                                       have been removed from the q because the current rcvd
                                       seg acked it, and the Complete routine above has taken
                                       it off the queue. */
460   3       DECLARE
                  fsn$rbs$p   POINTER,
                  fsn$rbs BASED fsn$rbs$p
          $IF f7
          $ELSE
          $SAVE NOLIST INCLUDE (:F1:TCLRBS.INC)
                  ;

461   3       IF c.cbtq$buf$cnt = 0 THEN      /* its case (3) - so highest sent must */
462   3           RETURN(c.highest$sent);     /* have been our FIN's seq no  */
                                              /* if we get here, its case (1) or (2). */
463   3       fsn$rbs$p = c.cbtq$hdr;         /* set up rbs ptr to search xmit q for Close req */
464   3       DO WHILE fsn$rbs.req <> close$req;
465   4           fsn$rbs$p = fsn$rbs.link;   /* not found, get next entry */
466   4       END;
467   3       RETURN(fsn$rbs.last$seq);       /* last.seq is always correct, even if data */
                                              /* is sent along with close req */
```

```
468   3       END finSseq$no;

              /*  --->>  Main Received Segment Handler Logic in Receive Process <<---  */

                                          /* when we get here, the cdb ptr has */
                                          /* been set up, and the received pkt */
                                          /* is in rp.    */
469   2       cur$cid = lcid$vector(cur$cdb$index);   /* set common variable */
470   2       IF c.state = listen THEN

              /* Listen */
                                          /* In the listen state we have previously done
                                          a passive open, so we are waiting for a SYN seg
                                          from some remote TCL client. */
471   2           DO;
472   3               IF rp_.ctl = syn$mask THEN
473   3                   DO;
474   4                       CALL accept$conn(synrcvd OR from$listen,irb$send$synack);
475   4                   END;
476   3               RETURN;                /* ignore any other packet - it can only be a
                                              delayed duplicate or an improper re-use of
                                              connection ID */
477   3           END; /* of listen */

478   2       IF c.state = synsent THEN

              /* Synsent */
                                          /* In the synsent state we have previously done
                                          an active open, so we are waiting for a SYN ACK
                                          (reply from remote passive open), an ACK (reply
                                          from remote active open whose own SYN hasn't
                                          gotten here yet, either due to his SYN seg being
                                          dropped, his SYN having been previously rejected
                                          by us [so he's in his timeout, waiting to re-try
                                          his SYN], or his SYN having been routed the long way
                                          around the net), or a SYN from remote active
                                          open, sent about the time ours was. */
479   2           DO;
                                          /* The following test is to fix up a simultaneous
                                          active open problem, and resolves it by allowing
                                          TCL to save the remote guy's ack no before we
                                          get his syn (which only happens because his Syn
                                          was lost, and we get his ack of our Syn first,
                                          then receive his syn on a re-transmission. */

                                          /* if it isn't an RST, save the ack num */
                                  /* ????????????? */
480   3               IF (rp_.ctl AND rstack$mask) = ack$mask THEN c.his$ack$no = rp_.seg$ack$no;

                                              /* deciding what to do with the received */
                                              /* segment ... first see if its a */
                                              /* response from a passive open cdb: */
482   3               IF ((rp_.ctl AND ctl$bits$mask) XOR synack$mask) = 0 THEN   /* syn,ack are only ctl bits on */
483   3                   DO;                    /* we got response to our syn- send */
484   4                       CALL accept$conn(estab, irb$send$flag); /* ack, enter estab state */
```

```
                                        /* adaptive retry timeout computation: */
485   4                 CALL set$retran$timeout;

486   4                 CALL chk$deferred$status(cur$cdb$p);    /* send back def stat RB, if any */
487   4                 CALL rp$err$flow;       /* do estab processing in case */
488   4                 RETURN;                 /* there's more stuff in packet */
489   4             END;
                                        /* ... not a pass opn resp, see if its */
                                        /* simultaneous active open: */
490   3             IF ((rp_.ctl AND ctl$bits$mask) XOR syn$mask) = 0 THEN
491   3                 DO;                   /* syn is only ctl bit on: */
492   4                 IF c.his$ack$no = 1 THEN
                                        /* my syn has been acked at some point */
493   4                     DO;
494   5                         CALL accept$conn(estab, irb$send$flag);
495   5                         CALL chk$deferred$status(cur$cdb$p);
496   5                     END;
497   4                 ELSE                  /* my syn hasn't been acked */
                            CALL accept$conn(synrcvd, irb$send$flag);
498   4                 RETURN;
499   4                 END;
                                        /* ... not a simultaneous active open, */
                                        /* see if its a reset: */
500   3             IF (rp_.ctl AND rst$mask) <> 0 THEN
501   3                 DO;                   /* my conn req was rejected */
502   4                 CALL clear$cdb$alarms(cur$cdb$p);   /*  kill timers */
503   4                 IF (c.persist$cnt:=c.persist$cnt+1) <= c.persist THEN
                                        /* persist count not expired, so just */
                                        /* ignore the refusal, and schedule */
504   4                     DO;               /* another SYN to be sent soon. */
505   5                         c.ctl$acb$irb$type = irb$sendsyn OR irb$timeout$mask
                                                   OR irb$ctlacb$mask;
506   5                         CALL cq$set$alarm(@c.ctl$alarm$cb, .tp$mbx,
                                    high(c.retran$to$dw), low(c.retran$to$dw));
507   5                     END;
508   4                 ELSE
                            DO;               /* persist count expirdw), give up */
509   5                         CALL try$to$delete$cdb(cur$cdb$index, cur$cdb$p, rem$abort);    /* set state, clr
            -lists, */
                                                           /* clr alarms */
        $IF log
        $ENDIF
510   5                     END;
511   4                 RETURN;
512   4             END;
                                        /* not a reset- see if its an ack of my syn
                                        (implying that remote guy is doing simulta-
                                        neous open, but his SYN hasn't arrived yet) */
513   3             IF rp_has_ack THEN
514   3                 DO;               /* there's an ack - if it appears to be for
                                        current connection, save all the info about
                                        remote guy. Note: using the accept$conn
                                        routine to do this, even though I don't
                                        want TP to send anything now. */
515   4                 IF (rp_.seg$seq$no = 1) AND (rp_.seg$ack$no = 1) THEN
516   4                     DO;
517   5                         CALL accept$conn(synsent, irb$null);
```

```
518   5                         CALL set$retran$timeout;
519   5                     END;
520   4                 ELSE CALL send$rst$reply(rst$old$dupl);
521   4                 RETURN;
522   4             END;

523   3         RETURN;
524   3     END;    /* of synsent */

        /* synrcvd, estab, finwait-1, finwait-2,
           time-wait, close-wait, closing, closed  */

           /* Filter out old duplicate segment sequence numbers that have been
           acknowledged, except that we must accept ACK control segments which
           are duplicates of the last segment, since they may carry new ACK and
           window values.  Note that if remote guy is only receiving data, he
           may never generate a new sequence number until he sends FIN. */

           /* There are therefore two tests for an acceptable packet at this stage:

           Case (1): Lone ACKs: if no data in seg and no sequence-consuming controls
                   (i.e., syn,fin,rst,eom), only ack, the test is that the segment
                   sequence number is not older than the last one we acked:
                    should have ...  rp_.seg$seq$no >= c.my$ack$no

           Case (2): Otherwise (i.e., length <> 0 or other controls present), the test
                   is that the segment sequence number should be newer than the last
                   one we acked: should have ...  rp_.seg$seq$no > c.my$ack$no
                   (All comparisons modulo 64K, of course, to account for sequence number
                    wrap-around)  */

                      /* Note: this test deliberately lets thru out-of-sequence segments
                      (i.e., in the window, but some earlier ones have not been received)
                      in order to allow the re-assembly routine the freedom to use
                      re-assembly buffers if it ever gets any ! */

525   2     IF rp_.source$cid <> c.rem$cid THEN /* Its not for this connection, so reject it */
526   2         DO;
527   3             CALL send$rst$reply(rst$old$dupl);
528   3             RETURN;
529   3         END;
                                            /* if remote guy sent an Are-You-There */
                                            /* signal, force a reply to be sent */
530   2     IF (rp_.ctl AND ayt$mask) <> 0 THEN CALL rp$defer$irb$tp(irb$send$flag);


532   2     IF (rp_.seg$data$len = 0)           /* no client data */
              AND (((rp_.ctl AND ctl$bits$mask) XOR ack$mask) = 0)    /* only ack bit on */ THEN
533   2         DO;                             /* Its case 1 */
534   3             IF ge$mod64k( rp_.seg$seq$no, c.my$ack$no )
535   3                 THEN GOTO accept$packet;
536   3                 ELSE GOTO reject$packet;
537   3         END;
538   2     ELSE                                /* Its case 2 */
              DO;
539   3             IF gt$mod64k( rp_.seg$seq$no, c.my$ack$no )
```

ayt ?

```
540   3                 THEN GOTO accept$packet;
541   3                 ELSE GOTO reject$packet;
542   3          END;

543   2      reject$packet:                        /* failed the sequence number filter - */
                                                   /* send an ACK segment with correct */
                                                   /* seq no, ack no, window */

             $IF log
             $ENDIF


             CALL rp$defer$irb$tp(irb$send$flag);
                                                   /* Note pkt reject in conn and TCL total ctrs */
544   2      CALL stky_incr(@c.pkts$rej);
545   2      CALL stky_incr(@tot$pkts$rej);
546   2      RETURN;

             /* If we get this far, packet has passed CID and sequence number tests */

547   2      accept$packet:

             /****CALL log$event(log$disp$wor$text,1,@cur$cid,36,
                              @('Packet Accepted, CDB values follow: '));****/
             /****CALL log$event(log$dwt,11,@c.my$ack$no,SIZE(cdb$label$string),
                                                   @cdb$label$string);****/

             /* Now process the received packet according to connection state */
             DO CASE(c.state AND state$mask);
             /* Listen - can't get here */
548   3          ;
             /* Syn sent - can't get here */
549   3          ;
             /* Syn Received */
550   3          DO;
551   4              IF rp_has_ack THEN
552   4                  DO;                      /* Only pkts with ACK are meaningful at this point - */
                                                   /* duplicates, etc were filtered out above */
553   5                      IF rp_.seg$ack$no = 1 THEN  /* he acked our syn,ack */
554   5                          DO;
555   6                              IF (rp_.ctl AND rst$mask) = 0 THEN  /* Its not an rst, */
556   6                                  DO;                             /* so our syn,ack was acknowledged - */
                                                   /* adaptive retry timeout computation: */
557   7                                      CALL set$retran$timeout;

558   7                                      CALL cq$clear$alarm(@c.data$alarm$cb);
559   7                                      c.state = estab;        /* and enter Estab State */
             $IF log
             $ENDIF
560   7                                      c.no$confid = 0;        /* reset retry count */
561   7                                      c.cum$retran$dw = 0;    /* clear the cummulative retry time */
562   7                                      CALL chk$deferred$status(cur$cdb$p);
563   7                                      CALL rp$err$flow;       /* do estab processing */

                                                   /*  Now do FIN processing, in case a */
                                                   /* FIN accompanies the ACK (could be */
                                                   /* because remote client submitted */
```

```
                                        /* data/close before the last ack of */
                                        /* connect handshake went out, or */
                                        /* because the first ack was lost, and */
                                        /* the close was submitted before the */
                                        /* re-xmit, or because the other TCL */
                                        /* allows close before estab, but queues */
                                        /* it. */
564   7                                 IF rp_has_fin THEN
                                        /* We got Close request from remote client */
565   7                                 DO;
566   8                                     IF rp_.seg$seq$no = c.my$ack$no THEN    /* put routine acked it */
567   8                                     DO;
568   9                                         c.state = clswait;      /* enter close-wait state */
      $IF log
      $ENDIF
569   9                                     END;
570   8                                 END;

571   7                             END;
572   6                         ELSE
                                DO;     /* legal RST received-abort */
573   7                             CALL clear$cdb$alarms(cur$cdb$p);   /* kill all timers on this cdb*/
574   7                             IF (c.state AND from$listen) <> 0 THEN
575   7                             DO;         /* came from listen-return to it */
                                                  /* put this cdb back on match list */
576   8                                 spec$type(cur$cdb$index) = spec$type(cur$cdb$index) AND 7FH;
577   8                                 c.next$transmit = 1;
578   8                                 c.state = listen;
579   8                                 c.cum$retran$dw = 0;    /* clear the cummulative retry time */
580   8                                 c.rem$cid = 0;          /* clr rem cid so it won't match on duplic
          -ate chk */
581   8                                 c.highest$sent = 0;
582   8                                 c.seen = 0;
583   8                                 c.his$ack$no = 0;
584   8                                 c.my$ack$no = 0;
585   8                                 c.timed$seq$no = 0;
586   8                                 c.send$flag = 0;
587   8                                 c.no$confid = 0;
      $IF log
      $ENDIF
588   8                                 RETURN;
589   8                             END;
590   7                         ELSE
                                DO;     /* came from synsent-abort */
                                                  /* set state, clr lists, clr alarms */
591   8                             CALL try$to$delete$cdb(cur$cdb$index, cur$cdb$p, rem$abort);

      $IF log
      $ENDIF
592   8                             RETURN;
593   8                         END;
594   7                     END;
595   6                 END;
596   5             ELSE                /* ack value was bad-send rst */
                        CALL send$rst$reply(rst$illegal$ack);
597   5             END;
```

```
598  4          END;    /* of syn received processing */


           /* Established */
599  3       IF estab$rst$chk$false THEN
600  3           DO;
601  4               CALL rp$err$flow;
                                          /* do FIN processing */
602  4               IF rp_has_fin THEN
                                          /* We got Close request from remote client */
603  4                   DO;
604  5                       IF rp_.seg$seq$no = c.my$ack$no THEN    /* put routine acked it */
605  5                           DO;
606  6                               c.state = clswait;      /* enter close-wait state */
           $IF log
           $ENDIF
607  6                           END;
608  5                   END;
609  4           END;

           /* Fin Wait 1 */
610  3       IF estab$rst$chk$false THEN
611  3           DO;
612  4               CALL rp$err$flow;
613  4               IF rp_has_ack AND (rp_.seg$ack$no = fin$seq$no) THEN
614  4                   DO;                    /* our FIN has been acked */
615  5                       c.state = finwait$2;
           $IF log
           $ENDIF
616  5                       CALL rest$of$finwait$2;         /* shared code with finwait2 */
617  5                       RETURN;
618  5                   END;
619  4               IF rp_has_fin THEN          /* We received a FIN from remote guy */
620  4                   DO;
621  5                       IF rp_.seg$seq$no = c.my$ack$no THEN
622  5                           DO;
623  6                               c.state = closing;
           $IF log
           $ENDIF
624  6                           END;
625  5                   END;
626  4           END;

           /* Fin Wait 2 */
627  3       IF estab$rst$chk$false THEN
628  3           DO;
629  4               CALL rp$err$flow;
630  4               CALL rest$of$finwait$2;     /* its a subroutine so that finwait1 code can use it */
631  4           END;

           /* Time Wait */
632  3       DO;
633  4           scratch = estab$rst$chk$false;

                       /* Nothing to do - we're just keeping the conn open in case */
                       /* a FIN gets re-transmitted, so we can send an ACK in reply */
                       /* to it. */
```

```
634   4       END;

              /* Close Wait */
                        /* Waiting on local client close, after having rcvd and Acked
                        a FIN.  If the FIN that put us here is re-transmitted, the
                        pre-filter will catch it and send an ack */
635   3           IF estab$rst$chk$false THEN CALL rp$err$flow;

              /* Closing */
637   3       DO;
638   4           IF estab$rst$chk$false THEN
639   4           DO;
640   5               CALL rp$err$flow;
641   5               IF rp_has_ack AND (rp_.seg$ack$no = fin$seq$no)  THEN
                                              /* Our Fin was acked- we can */
642   5                   DO;                   /* enter closed to wait for ackrst  */
643   6                   CALL try$to$delete$cdb(cur$cdb$index, cur$cdb$p, ok$closed$resp);   /* set state, clr list
                    -s, */
                                              /* clr alarms */
              $IF log
              $ENDIF
644   6                   END;
645   5           END;
646   4       END;

              /* Closed */
647   3           CALL send$rst$reply(rst$conn$closed);       /* cdb closed-so */
                                              /* unless pkt is RST, send */
                                              /* back an RST */
648   3       END; /* of case */


649   2       END process$segment;

                                                              /**********************/
                                                              /**  self$connect    **/
                                                              /**********************/
650   1       self$connect: PROCEDURE BYTE;
                                              /* Subroutine for connect match and SYN */
                                              /* processing -- detects case in which */
                                              /* a loop-back pkt has matched its */
                                              /* own connection data base in order to */
                                              /* disallow the match. Returns "true" if */
                                              /* self-connect match is detected. */
651   2           IF (rp_.source$cid = c.loc$cid)     AND
                      dlsource_eq_host(@loc$host(0)) THEN
652   2               RETURN(true);/* trying to connect to self */
653   2           ELSE RETURN (false);
654   2       END self$connect;

                                                              /**********************/
                                                              /**  connect$match   **/
                                                              /**********************/
655   1       connect$match: PROCEDURE(select$case) BYTE;
656   2       DECLARE select$case BYTE;           /* selects the type of test */
```

```
                        /* main-line code for connect match */
                                                /* chk locport vs destport for all cases */

657   2       IF c.loc$port <> rp_.dest$port THEN RETURN(false);

659   2       DO CASE(select$case);
              /* Case 0: Fully specified Open, active or passive */

660   3           IF (c.rem$port = rp_.source$port)    AND
                    dlsource_eq_host(@c.rem$host(0) ) THEN
661   3            DO;
662   4               IF self$connect THEN RETURN(false);     /* matched myself */
664   4               CALL process$segment;
665   4               RETURN(true);           /* tell main loop we got a match */
666   4            END;
667   3           ELSE RETURN(false);

              /* Case 1: Partially specified Passive Open (any remote host, specified port) */

668   3           IF (c.rem$port = rp_.source$port) THEN
669   3            DO;
670   4               IF self$connect THEN RETURN(false);     /* matched myself */
672   4               CALL process$segment;
673   4               RETURN(true);           /* tell main loop we got a match */
674   4            END;
675   3           ELSE RETURN(false);

              /* Case 2: Unspecified Passive Open (any remote host, any port) */

676   3           DO;
677   4               IF self$connect THEN RETURN(false);     /* matched myself */
679   4               CALL process$segment;
680   4               RETURN(true);           /* tell main loop we got a match */
681   4           END;

682   3       END;    /* of do case */

683   2       END connect$match;

                                                        /*********************/
                                                        /****  rcv$proc  ****/
                                                        /*********************/
684   1       rcv$proc: PROCEDURE PUBLIC;

                        /* This is TCL's Receive Process (RP).  It is declared to KAOS
                        statically in the TCL KAOS objects macro file, called
                        TCLGNL.A86, orTCLGEN.A86, or even EPGEN.A86, depending
                        on the configuration being generated.

                        RP is the only thing in TCL that receives segments from Data Link;
                        it always gives them back directly to Data Link (receive seg
                        buffers are not held or sent to othe processes).  The job of RP
                        is to receive and process segments, and to update variables in the
                        connection data base to indicate the progress of transmitted and
                        received data and TCL connection states.  RP does not send
```

```
                segments directly; it sends short messages (called Internal
                Request Blocks, or IRBs) to the Transmit Process when a segment
                needs to be sent (an ACK of data just received, for example).
                To cause a Reset (RST) segment to be sent RP sends an LIRB
                (Long IRB, so named because it holds more info than an IRB) to TP.
                The IRBs and LIRBs are obtained from free buffer lists called
                "free$irb$mbx" and "free$lirb$mbx", organized as KAOS mailboxes,
                and declared to KAOS the same way as RP, above.  IRBs and LIRBs
                are sent only to TP, which is the only consumer of same, and who
                replaces them on the proper free buffer list.

                RP operates roughly as follows: wait for a seg to arrive, then
                acquire the TCL scheduler lock semaphore, to insure non-interference
                from the other TCL processes.  Validate the checksum and version
                number, then do the real processing:
                * for segs other than the initial SYN (assumed to be the "normal"
                case), try to match the received seg up to its connection data base
                by checking the destination Connection ID against the list of
                active CIDs in this node ("lcid$vector", declared in TSTART).
                Assuming there's a match, it calls process$segment, which implements
                the basic connection state machine, and eventually (once the
                connection is established), calls the basic error and flow control
                subroutine.
                * for SYN-only segments, we first have to see if the seg is a dupliacte
                of a previous seg, then if it isn't, we try to match it up with
                an open request, if any.  We must attempt to match the most
                completely specified open requests first, then partially specified,
                then unspecified.  Note that it is necessary to check that the
                incoming request isn't matching its own connection data base
                for segments sent from and addressed to this node.  Once we
                have a match, process$segment is called to move thru the connection
                state machine. */

685   2       DO forever;
686   3       wait$for$segment:
                    rp$p = cq$receive(.rp$mbx);

687   3           CALL cq$waitsem(.sched$lock);   /* acquire the cdb lock before proceding */

                $IF dbg
                $ENDIF
                $IF log
                $ENDIF
                                                /* make a non-based local copy of received
                                                segment header for code size and avoiding
                                                contention in static RAM. */
688   3           CALL MOVW(@rp.dl$source, @rp_.dl$source0, 14);
                                                /* set up booleans for frequent tests */
689   3           rp_has_ack = (rp_.ctl AND ack$mask) <> 0;
690   3           rp_has_fin = (rp_.ctl AND fin$mask) <> 0;

691   3           IF chk$sum$calc(rp$o) <> rp_.checksum THEN
692   3               DO;                        /* checksum calculation failed: */
693   4                   CALL stky_incr(@bad$chk$sum);
                $IF log
                $ENDIF
```

```
694   4                 GOTO exit$block;        /* throw away seg and give up semaphore */
695   4             END;

                                                /* Test to see that version number of the
                                                sending TCL is compatible - The lower order
                                                byte may differ.  Hence 101h will talk to
                                                104h */
696   3         IF (rp_.tcl$version AND 0FF00H) <> (tcl$version AND 0FF00H) THEN
697   3             DO;
698   4                 CALL send$rst$reply(rst$version$mismatch);
699   4                 GOTO exit$block;
700   4             END;

                                                /* preliminary validation done, now figure */
                                                /* out what to do with the received seg. */
701   3         IF rp_.ctl <> syn$mask THEN     /* Its not a lone SYN, so its probably in reply  */
702   3             DO;                         /* to something we sent - check dest cid */
703   4                 IF rp_.dest$cid <> 0 THEN   /* validate CID if its legal */
704   4                     DO;
705   5                         IF (cur$cdb$index:=search_lcid$vector(rp_.dest$cid)) <> 0FFFFH THEN
706   5                             DO;         /* dest cid present, chk for match */
707   6                                 CALL setup$cdb(cur$cdb$index, .cur$cdb$p);
708   6                                 IF (rp_.source$port = c.rem$port) AND
                                              (rp_.dest$port = c.loc$port) THEN
709   6                                     CALL process$segment;   /* a match to existing cid  */
710   6                                 ELSE CALL send$rst$reply(rst$old$dupl);
711   6                             END;
712   5                         ELSE            /* CID is not at this node */
                                      CALL send$rst$reply(rst$no$match);
713   5                     END;
714   4                 ELSE CALL send$rst$reply(rst$zero$dest$cid);/* Dest cid=0 is illegal for non-SYN */
715   4             END;    /* of non-SYN processing */

716   3         ELSE                            /* Only SYN control bit is set, so */
                                                /* Its a connect request: first, see if */
                                                /* its a duplicate of an earlier SYN */
                  DO;
717   4                 DO cur$cdb$index = 0 TO cur$max$cdbs-1;          /* Check every */
718   5                     IF lcid$vector(cur$cdb$index) <> 0 THEN      /*  allocated cdb */
719   5                         DO;
720   6                             CALL setup$cdb(cur$cdb$index, .cur$cdb$p);
721   6                             IF (c.rem$cid = rp_.source$cid) AND
                                          dlsource_eq_host(@c.rem$host(0) ) AND
                                          NOT self$connect THEN
722   6                                 DO;
723   7                                     CALL process$segment;
724   7                                     GOTO exit$block;
725   7                                 END;
726   6                         END;
727   5                 END; /* of iterative do */
                                                /* Wasn't a duplicate SYN, so try to */
                                                /* match it with a pending Open req */
                                                /* go thru all cdb's up to 3 times, with */
                                                /*  matching criteria successively less strict */
728   4                 DO match$case = 0 TO 2; /* matchcase=0=>fully spec'd, =1=> partially spec'd, */
                                                /* =2=> unspec'd */
                                                /* Check every allocated cdb that is in an */
```

```
                                        /* un-synchronized state */
729   5                 DO cur$cdb$index = 0 TO cur$max$cdbs-1;
730   6                     IF lcid$vector(cur$cdb$index) <> 0 AND
                                spec$type(cur$cdb$index) = match$case THEN  /* FFH or high-order */
                                                                            /* bit set indicates not in list */
731   6                         DO;         /* found allocated and unsynchronized cdb*/
732   7                             CALL setup$cdb(cur$cdb$index, .cur$cdb$p);
                                            /* cdb-try to match rcvd SYN */
733   7                             IF connect$match(match$case) THEN GOTO exit$block;
735   7                         END;
736   6                 END;
737   5             END;
                                        /* if we get here, there was no match */
738   4             CALL send$rst$reply(rst$syn$refused);
739   4         END;
740   3   exit$block: CALL cq$dll$rx$ret$buf(rp$o);   /* Give back the rcv pkt buffer */

        $IF dbg
        $ENDIF

741   3   CALL cq$signal(.sched$lock);
                                        /* now that we have released */
                                        /* sched lock, send any deferred */
                                        /* irb's requested this time thru */
742   3   CALL send$deferred$irbs (.rp$irb$index, .rp$irb$list, cur$cdb$index, cur$cid);

743   3   END; /* of forever loop */
744   2   END rcv$proc;
745   1   END rp;
```

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES		
12			ABORTREQ	LITERALLY '8'	
181	0094H	133	ACCEPTCONN	PROCEDURE STACK=0018H	474 484 494 497 517
547	09A2H		ACCEPTPACKET	LABEL IN PROC (PROCESSSEGMENT)	535 540
6			ACKMASK.	LITERALLY '1000H'	480 532 689
118	0000H	4	ALARMP	POINTER IN PROC (CQSETALARM) PARAMETER	118
124	0000H	4	ALARMP	POINTER IN PROC (CQCLEARALARM) PARAMETER	124
121	0000H	4	ALARMP	POINTER IN PROC (CQCHECKALARM) PARAMETER	121
115	0000H	4	ALARMP	POINTER IN PROC (CQCREATEALARM) PARAMETER	115
6			AYTMASK.	LITERALLY '2000H'	530
3	0000H	2	BADCHKSUM.	WORD EXTERNAL(13)	693
4	0000H	2	BUFMIPMBX.	WORD EXTERNAL(19)	252 349 442
131	0000H	2	BUFO	WORD IN PROC (CQDLLRXRETBUF) PARAMETER	131
12			BUFTOOSHORT.	LITERALLY '8'	
7	0000H	124	C.	STRUCTURE BASED(CURCDBP)	
	0000H	1	STATE.	BYTE	191* 452* 470 478 547 559* 568* 574 578* 606* 615* 623*
	0001H	1	OWNERDEVICE. . . .	BYTE	
	0002H	2	OWNERPROCESSID . .	WORD	
	0004H	2	LOCCID	WORD	651
	0006H	2	LOCPORT.	WORD	657 708
	0008H	2	REMNET	WORD	185*
	000AH	6	REMHOST.	WORD ARRAY(3)	186 660 721
	0010H	2	REMPORT.	WORD	187* 660 668 708
	0012H	2	PERSIST.	WORD	503
	0014H	2	ABORTTOHI.	WORD	
	0016H	2	REMCID	WORD	188* 525 580* 721
	0018H	4	RETRANTODW	DWORD	365* 369* 369 372* 372 373 375* 420 455 506
	001CH	2	RESERVED	WORD	
	001EH	2	TIMEDSEQNO	WORD	357* 379 585*
	0020H	4	SEGTRANSTIMEDW . .	DWORD	358
	0024H	4	CUMRETRANDW. . . .	DWORD	193* 423* 561* 579*
	0028H	2	PERSISTCNT	WORD	503* 503
	002AH	4	CBTQHDR.	POINTER	339 347* 352 463
	002EH	4	PCBQHDR.	POINTER	249* 257 299
	0032H	4	DEFSTATUSP	POINTER	
	0036H	2	MYACKNO.	WORD	190* 274 290* 318* 450 534 539 566 584* 604 621
	0038H	2	SEEN	WORD	582*
	003AH	1	MYCREDIT	BYTE	251* 251
	003BH	1	CURBLKINDEX. . . .	BYTE	231 234 235 256* 266* 266 267 301
	003CH	2	CBDATAINDEX. . . .	WORD	233* 306 309* 309
	003EH	2	RCVBYTESCONSUMED .	WORD	284 297 306 308 310* 310 317*
	0040H	2	CURBLKLENLEFT. . .	WORD	234* 237* 303 313* 313 320
	0042H	2	HISACKNO	WORD	405 407 410 412* 413 425 481* 492 583*
	0044H	2	NEXTTRANSMIT . . .	WORD	415* 415 577*
	0046H	1	CLOSEDREASON . . .	BYTE	
	0047H	1	HISCREDIT.	BYTE	189* 408 409* 425
	0048H	2	HIGHESTSENT. . . .	WORD	462 581*
	004AH	1	CBTQBUFCNT	BYTE	337 344* 350* 350 417 425 461
	004BH	1	PCBQBUFCNT	BYTE	253* 253 254 278
	004CH	2	PKTSREJ.	WORD	544

```
        004EH     2       PKTSRETRAN . . . .    WORD
        0050H     2       NOCONFID . . . . .    WORD        192*  379  383  392  399  422*  560*  587*
        0052H     2       LASTNOCONFID . . .    WORD        383  392  399*
        0054H     1       RETRANSMITSTATE. .    BYTE        376*  381  386*  394*
        0055H     1       SENDFLAG . . . . .    BYTE        586*
        0056H     2       PENDINGRCVDATA . .    WORD        285*  296*  329*
        0058H     2       RCVBUFREJCNT . . .    WORD        282
        005AH     2       AYTCOUNT . . . . .    WORD
        005CH     4       DATAALARMCB. . . .    WORD ARRAY(2)           416  420  558
        0060H     1       DATAACBIRBTYPE . .    BYTE        419*
        0061H     1       DATAACBFLAG. . . .    BYTE
        0062H    10       DATAACBREM . . . .    BYTE ARRAY(10)
        006CH     4       CTLALARMCB . . . .    WORD ARRAY(2)           455  506
        0070H     1       CTLACBIRBTYPE. . .    BYTE        454*  505*
        0071H     1       CTLACBFLAG . . . .    BYTE
        0072H    10       CTLACBREM. . . . .    BYTE ARRAY(10)
  229   0000H     1   CBDATA . . . . . .        BYTE BASED(CBDATAP) ARRAY(1) IN PROC (PUT)     306  308*
  229   003EH     4   CBDATAP. . . . . .        POINTER IN PROC (PUT)         235*  301*  306
  166   0000H   124   CCAC . . . . . .         STRUCTURE BASED(CCACDBP) IN PROC (CLEARCDBALARMS)
        0000H     1       STATE. . . . . .    BYTE
        0001H     1       OWNERDEVICE. . . .  BYTE
        0002H     2       OWNERPROCESSID . .  WORD
        0004H     2       LOCCID . . . . . .  WORD
        0006H     2       LOCPORT. . . . . .  WORD
        0008H     2       REMNET . . . . . .  WORD
        000AH     6       REMHOST. . . . . .  WORD ARRAY(3)
        0010H     2       REMPORT. . . . . .  WORD
        0012H     2       PERSIST. . . . . .  WORD
        0014H     2       ABORTTOHI. . . . .  WORD
        0016H     2       REMCID . . . . . .  WORD
        0018H     4       RETRANTODW . . . .  DWORD
        001CH     2       RESERVED . . . . .  WORD
        001EH     2       TIMEDSEQNO . . . .  WORD
        0020H     4       SEGTRANSTIMEDW . .  DWORD
        0024H     4       CUMRETRANDW. . . .  DWORD
        0028H     2       PERSISTCNT . . . .  WORD
        002AH     4       CBTQHDR. . . . . .  POINTER
        002EH     4       PCBQHDR. . . . . .  POINTER
        0032H     4       DEFSTATUSP . . . .  POINTER
        0036H     2       MYACKNO. . . . . .  WORD
        0038H     2       SEEN . . . . . .    WORD
        003AH     1       MYCREDIT . . . . .  BYTE
        003BH     1       CURBLKINDEX. . . .  BYTE
        003CH     2       CBDATAINDEX. . . .  WORD
        003EH     2       RCVBYTESCONSUMED .  WORD
        0040H     2       CURBLKLENLEFT. . .  WORD
        0042H     2       HISACKNO . . . . .  WORD
        0044H     2       NEXTTRANSMIT . . .  WORD
        0046H     1       CLOSEDREASON . . .  BYTE
        0047H     1       HISCREDIT. . . . .  BYTE
        0048H     2       HIGHESTSENT. . . .  WORD
        004AH     1       CBTQBUFCNT . . . .  BYTE
        0048H     1       PCBQBUFCNT . . . .  BYTE
        004CH     2       PKTSREJ. . . . . .  WORD
        004EH     2       PKTSRETRAN . . . .  WORD
        0050H     2       NOCONFID . . . . .  WORD
```

```
      0052H    2      LASTNOCONFID . . .   WORD
      0054H    1      RETRANSMITSTATE. .   BYTE
      0055H    1      SENDFLAG . . . . .   BYTE
      0056H    2      PENDINGRCVDATA . .   WORD
      0058H    2      RCVBUFREJCNT . . .   WORD
      005AH    2      AYTCOUNT . . . . .   WORD
      005CH    4      DATAALARMCB. . . .   WORD ARRAY(2)          167
      0060H    1      DATAACBIRBTYPE . .   BYTE
      0061H    1      DATAACBFLAG. . . .   BYTE       169
      0062H   10      DATAACBREM . . . .   BYTE ARRAY(10)
      006CH    4      CTLALARMCB . . . .   WORD ARRAY(2)          168
      0070H    1      CTLACBIRBTYPE. . .   BYTE
      0071H    1      CTLACBFLAG . . . .   BYTE       169
      0072H   10      CTLACBREM. . . . .   BYTE ARRAY(10)
  166 0004H    4  CCACDBP. . . . . .       POINTER IN PROC (CLEARCDBALARMS) PARAMETER AUTOMATIC      166  167
                                            168  169
   18 0000H    1  CDBINDEX . . . . . .     BYTE IN PROC (DELETECDB) PARAMETER        18
   51 0000H    1  CDBINDEX . . . . . .     BYTE IN PROC (SENDDEFERREDIRBS) PARAMETER       51
   18 0000H    4  CDBP . . . . . . . .     POINTER IN PROC (DELETECDB) PARAMETER           18
   30 0000H    4  CDBP . . . . . . . .     POINTER IN PROC (GETSTATUSINFO) PARAMETER       30
   54 0000H    4  CDBP . . . . . . . .     POINTER IN PROC (DEFERIRBTP) PARAMETER          54
   21 0000H    4  CDBP . . . . . . . .     POINTER IN PROC (CLEARLISTS) PARAMETER          21
   27 0000H    4  CDBP . . . . . . . .     POINTER IN PROC (CHECKAYTTIMER) PARAMETER       27
   15 0000H    2  CDBPO. . . . . . . .     WORD IN PROC (SETUPCDB) PARAMETER        15
    3 005FH    1  CDBSTRIED. . . . . .     BYTE
  434 0000H  124  CDSC . . . . . . . .     STRUCTURE BASED(CDSCDBP) IN PROC (CHKDEFERREDSTATUS)
      0000H    1      STATE. . . . . .     BYTE
      0001H    1      OWNERDEVICE. . . .   BYTE
      0002H    2      OWNERPROCESSID . .   WORD
      0004H    2      LOCCID . . . . . .   WORD
      0006H    2      LOCPORT. . . . . .   WORD
      0008H    2      REMNET . . . . . .   WORD
      000AH    6      REMHOST. . . . . .   WORD ARRAY(3)
      0010H    2      REMPORT. . . . . .   WORD
      0012H    2      PERSIST. . . . . .   WORD
      0014H    2      ABORTTOHI. . . . .   WORD
      0016H    2      REMCID . . . . . .   WORD
      0018H    4      RETRANTODW . . . .   DWORD
      001CH    2      RESERVED . . . . .   WORD
      001EH    2      TIMEDSEQNO . . . .   WORD
      0020H    4      SEGTRANSTIMEDW . .   DWORD
      0024H    4      CUMRETRANDW. . . .   DWORD
      0028H    2      PERSISTCNT . . . .   WORD
      002AH    4      CBTQHDR. . . . . .   POINTER
      002EH    4      PCBQHDR. . . . . .   POINTER
      0032H    4      DEFSTATUSP . . . .   POINTER          435  437*  443
      0036H    2      MYACKNO. . . . . .   WORD
      0038H    2      SEEN . . . . . .     WORD
      003AH    1      MYCREDIT . . . . .   BYTE
      003BH    1      CURBLKINDEX. . . .   BYTE
      003CH    2      CBDATAINDEX. . . .   WORD
      003EH    2      RCVBYTESCONSUMED .   WORD
      0040H    2      CURBLKLENLEFT. . .   WORD
      0042H    2      HISACKNO . . . . .   WORD
      0044H    2      NEXTTRANSMIT . . .   WORD
      0046H    1      CLOSEDREASON . . .   BYTE
```

```
        0047H     1      HISCREDIT. . . . .    BYTE
        0048H     2      HIGHESTSENT. . . .    WORD
        004AH     1      CBTQBUFCNT . . . .    BYTE
        004BH     1      PCBQBUFCNT . . . .    BYTE
        004CH     2      PKTSREJ. . . . . .    WORD
        004EH     2      PKTSRETRAN . . . .    WORD
        0050H     2      NOCONFID . . . . .    WORD
        0052H     2      LASTNOCONFID . . .    WORD
        0054H     1      RETRANSMITSTATE. .    BYTE
        0055H     1      SENDFLAG . . . . .    BYTE
        0056H     2      PENDINGRCVDATA . .    WORD
        0058H     2      RCVBUFREJCNT . . .    WORD
        005AH     2      AYTCOUNT . . . . .    WORD
        005CH     4      DATAALARMCB. . . .    WORD ARRAY(2)
        0060H     1      DATAACBIRBTYPE . .    BYTE
        0061H     1      DATAACBFLAG. . . .    BYTE
        0062H    10      DATAACBREM . . . .    BYTE ARRAY(10)
        006CH     4      CTLALARMCB . . . .    WORD ARRAY(2)
        0070H     1      CTLACBIRBTYPE. . .    BYTE
        0071H     1      CTLACBFLAG . . . .    BYTE
        0072H    10      CTLACBREM. . . . .    BYTE ARRAY(10)
  434   0004H     4  CDSCDBP. . . . . .       POINTER IN PROC (CHKDEFERREDSTATUS) PARAMETER AUTOMATIC          434
                                                435  441  443
   26   0000H        CHECKAYTTIMER. . . .     PROCEDURE EXTERNAL(25) STACK=0000H       431
  433   07A1H   116  CHKDEFERREDSTATUS. . .   PROCEDURE PUBLIC STACK=0010H      486  495  562
   23   0000H        CHKSUMCALC . . . . .     PROCEDURE WORD EXTERNAL(24) STACK=0000H       691
   51   0000H     2  CID. . . . . . . . .    WORD IN PROC (SENDDEFERREDIRBS) PARAMETER      51
    6              CKSMASK. . . . . . .     LITERALLY '4000H'
  165   001BH    53  CLEARCDBALARMS . . .     PROCEDURE PUBLIC STACK=000CH      177  184  453  502  573
   20   0000H        CLEARLISTS . . . . .     PROCEDURE BYTE EXTERNAL(23) STACK=0000H       178
    8              CLOSED . . . . . . .     LITERALLY '9'           174
   12              CLOSEREQ . . . . . .     LITERALLY '2'           342  464
    8              CLOSING. . . . . . .     LITERALLY '8'           623
    8              CLSWAIT. . . . . . .     LITERALLY '7'           568  606
  335   04A8H   142  COMPLETE . . . . . .     PROCEDURE STACK=000CH        413
  655   0BEBH    97  CONNECTMATCH . . . .     PROCEDURE BYTE STACK=0026H         733
  229   0042H     2  COPYLEN. . . . . . .     WORD IN PROC (PUT)     303*  304  306  307  308  309  310  311
                                                312  313
  120   0000H        CQCHECKALARM . . . .     PROCEDURE BYTE EXTERNAL(57) STACK=0000H
  123   0000H        CQCLEARALARM . . . .     PROCEDURE EXTERNAL(58) STACK=0000H      167  168  416  558
  114   0000H        CQCREATEALARM. . . .     PROCEDURE EXTERNAL(55) STACK=0000H
   61   0000H        CQCREATELIST . . . .     PROCEDURE EXTERNAL(37) STACK=0000H
   84   0000H        CQCREATEMAILBOX. . .     PROCEDURE EXTERNAL(45) STACK=0000H
   69   0000H        CQCREATEPROCESS. . .     PROCEDURE EXTERNAL(40) STACK=0000H
   72   0000H        CQCREATESEMAPHORE. .     PROCEDURE EXTERNAL(41) STACK=0000H
   93   0000H        CQCRECEIVE . . . . .     PROCEDURE POINTER EXTERNAL(48) STACK=0000H
   81   0000H        CQCWAIT. . . . . . .     PROCEDURE BYTE EXTERNAL(44) STACK=0000H
  141   0000H        CQDLLCONNECT . . . .     PROCEDURE BYTE EXTERNAL(65) STACK=0000H
  135   0000H        CQDLLREAD. . . . . .     PROCEDURE WORD EXTERNAL(63) STACK=0000H
  138   0000H        CQDLLREADC . . . . .     PROCEDURE WORD EXTERNAL(64) STACK=0000H
  130   0000H        CQDLLRXRETBUF. . . .     PROCEDURE EXTERNAL(61) STACK=0000H      740
  133   0000H        CQDLLSTART . . . . .     PROCEDURE EXTERNAL(62) STACK=0000H
  126   0000H     2  CQDLLTXFREEMBX . . .    WORD EXTERNAL(59)
  127   0000H        CQDLLTXSEND. . . . .     PROCEDURE EXTERNAL(60) STACK=0000H
   66   0000H        CQHALTANDCATCHFIRE .     PROCEDURE EXTERNAL(39) STACK=0000H      170
  108   0000H        CQICRECEIVE. . . . .     PROCEDURE POINTER EXTERNAL(53) STACK=0000H
```

```
  102  0000H        CQICWAIT . . . . . .  PROCEDURE BYTE EXTERNAL(51) STACK=0000H
  105  0000H        CQISEND. . . . . .    PROCEDURE EXTERNAL(52) STACK=0000H
   99  0000H        CQISIGNAL. . . . . .  PROCEDURE EXTERNAL(50) STACK=0000H
  147  0000H        CQMIPCONNECT . . . .  PROCEDURE BYTE EXTERNAL(67) STACK=0000H
  153  0000H        CQMIPGETADDRESS. . .  PROCEDURE POINTER EXTERNAL(69) STACK=0000H     235  301
  156  0000H        CQMIPGETMIPFORM. . .  PROCEDURE POINTER EXTERNAL(70) STACK=0000H
  150  0000H        CQMIPREGISTER. . . .  PROCEDURE BYTE EXTERNAL(68) STACK=0000H
  144  0000H        CQMIPSEND. . . . . .  PROCEDURE BYTE EXTERNAL(66) STACK=0000H
   96  0000H        CQMRECEIVE . . . . .  PROCEDURE POINTER EXTERNAL(49) STACK=0000H
  111  0000H        CQREADCLOCK. . . . .  PROCEDURE EXTERNAL(54) STACK=0000H       356
   90  0000H        CQRECEIVE. . . . . .  PROCEDURE POINTER EXTERNAL(47) STACK=0000H     205  686
   64  0000H        CQSCHEDULE . . . . .  PROCEDURE EXTERNAL(38) STACK=0000H
   87  0000H        CQSEND . . . . . .    PROCEDURE EXTERNAL(46) STACK=0000H       218  252  349  442
  117  0000H        CQSETALARM . . . . .  PROCEDURE EXTERNAL(56) STACK=0000H       420  455  506
   75  0000H        CQSIGNAL . . . . . .  PROCEDURE EXTERNAL(42) STACK=0000H       204  741
   59  0000H        CQSTART. . . . . .    PROCEDURE EXTERNAL(36) STACK=0000H
   78  0000H        CQWAITSEM. . . . . .  PROCEDURE EXTERNAL(43) STACK=0000H       206  687
    6                CREDITMASK . . . . .  LITERALLY '003FH'      189  409
    6                CTLBITSMASK. . . . .  LITERALLY '1F00H'      482  490  532
  229  0069H     1  CTLBITSPRESENT . . .  BYTE IN PROC (PUT)     268  298*  322
    3  005EH     1  CURCDBINDEX. . . . .  BYTE        194  223  469  509  576  591  643  705*  707  717*
                                           717  718  720  727  729*  729  730  732  736  742
    7  002AH     4  CURCDBP. . . . . .    POINTER          163  184  186  223  231  234  235  251  253
                                           254  257  266  267  274  278  282  284  297  299  301
                                           303  306  308  309  310  313  320  337  339  350  352
                                           358  369  372  373  379  381  383  392  399  405  407
                                           408  410  413  415  416  417  420  425  431  450  453
                                           455  461  462  463  470  478  486  492  495  502  503
                                           506  509  525  534  539  544  547  558  562  566  573
                                           574  591  604  621  643  651  657  660  668  707  708
                                           720  721  732
    3  0000H     2  CURCID . . . . . .    WORD        469*  742
    3  0000H     1  CURMAXCDBS . . . . .  BYTE EXTERNAL(3)       717  729
  112  0000H     2  DATARTNO . . . . . .  WORD IN PROC (CQREADCLOCK) PARAMETER     112
  371  0653H    76  DECREASERET. . . . .  PROCEDURE IN PROC (UPDATERETRANTIMEOUT) STACK=0006H        387  388
                                           397
   53  0000H        DEFERIRBTP . . . . .  PROCEDURE EXTERNAL(34) STACK=0000H       163
    2                DEFNETIDLIT. . . . .  LITERALLY '1'
   12                DEFSTATUSREQ . . . .  LITERALLY '4'
  118  0000H     2  DELAYH . . . . . .    WORD IN PROC (CQSETALARM) PARAMETER     118
  118  0000H     2  DELAYL . . . . . .    WORD IN PROC (CQSETALARM) PARAMETER     118
   17  0000H        DELETECDB. . . . . .  PROCEDURE EXTERNAL(22) STACK=0000H       179
    2                DLLHEADERLEN . . . .  LITERALLY '14'
   47  0000H        DLSOURCE_EQ_HOST . .  PROCEDURE BYTE EXTERNAL(32) STACK=0000H        651  660  721
   70  0000H     2  ENTRYO . . . . . .    WORD IN PROC (CQCREATEPROCESS) PARAMETER      70
    6                EOMACKMASK . . . . .  LITERALLY '1200H'
    6                EOMFINMASK . . . . .  LITERALLY '0600H'      298  429
    6                EOMMASK. . . . . .    LITERALLY '0200H'      244  280
   67  0000H     2  ERRORCODE. . . . . .  WORD IN PROC (CQHALTANDCATCHFIRE) PARAMETER          67
    8                ESTAB. . . . . .      LITERALLY '3'          484  494  559
  220  0197H    35  ESTABRSTCHKFALSE . .  PROCEDURE BYTE STACK=001CH          599  610  627  633  635  638

  740  0DC5H        EXITBLOCK. . . . . .  LABEL IN PROC (RCVPROC)      694  699  724  734
    2                FALSE. . . . . .      LITERALLY '0'          224  262  327  653  658  663  667  671
                                           675  678
    6                FINACKMASK . . . . .  LITERALLY '1400H'
```

```
  6                 FINMASK. . . . . .   LITERALLY '0400H'      690
459  0B88H    62    FINSEQNO . . . . . .   PROCEDURE WORD IN PROC (PROCESSSEGMENT) STACK=0002H       613  641

  8                 FINWAIT1 . . . . . .   LITERALLY '4'
  8                 FINWAIT2 . . . . . .   LITERALLY '5'          615
  2                 FOREVER. . . . . .   LITERALLY 'WHILE true'       302  685
  4  0000H      2   FREEIRBMBX . . . . .   WORD EXTERNAL(17)
  4  0000H      2   FREELIRBMBX. . . . .   WORD EXTERNAL(18)      205
  8                 FROMLISTEN . . . . .   LITERALLY '80H'        474  574
460  0000H     32   FSNRBS . . . . . .   STRUCTURE BASED(FSNRBSP) IN PROC (FINSEQNO)
       0000H      1       CONTENTS . . . .   BYTE
       0001H      1       CREDIT . . . . .   BYTE
       0002H      2       LASTSEQ. . . . .   WORD       467
       0004H      4       MIPBUFBASE . . .   POINTER
       0008H      2       MIPLENGTH. . . .   WORD
       000AH      1       MIPIDSID . . . .   BYTE
       000BH      1       MIPOWNERDEVID. .   BYTE
       000CH      2       INTERNALPROCESSID. WORD
       000EH      1       REQ. . . . . .   BYTE       464
       000FH      1       RESP . . . . . .   BYTE
       0010H      2       RTNMIPSKT. . . .   WORD
       0012H      4       LINK . . . . . .   POINTER          465
       0016H      2       CID. . . . . .   WORD
       0018H      2       FIRSTSEQ . . . .   WORD
       001AH      2       CLIENTUSE. . . .   WORD
       001CH      2       BUFLEN . . . . .   WORD
       001EH      1       NUMBLKS. . . . .   BYTE
       001FH      1       VB . . . . . .   BYTE
460  005AH      4   FSNRBSP. . . . . .   POINTER IN PROC (FINSEQNO)          463*  464  465*  465  467
 38  0000H          GEMOD64K . . . . . .   PROCEDURE BYTE EXTERNAL(29) STACK=0000H          340  405  534
264  0477H     49   GETNEXTBLK . . . . .   PROCEDURE BYTE IN PROC (PUT) STACK=0014H          321  327
355  0536H     90   GETROUNDTRIP . . . . .   PROCEDURE STACK=0006H        364
 29  0000H          GETSTATUSINFO. . . . .   PROCEDURE BYTE EXTERNAL(26) STACK=0000H          441
 35  0000H          GTMOD64K . . . . . .   PROCEDURE BYTE EXTERNAL(28) STACK=0000H          410  539
159                 HACFCHKACB . . . . .   LITERALLY '438'
159                 HACFDLLCONN. . . . .   LITERALLY '401'
159                 HACFDLLREADHOST. . .   LITERALLY '402'
159                 HACFIPDEFIRB . . . .   LITERALLY '432'
159                 HACFMIPCONNECT . . .   LITERALLY '404'
159                 HACFMIPREGISTER. . .   LITERALLY '434'
159                 HACFMIPSEND. . . . .   LITERALLY '416'
159                 HACFPUTCNT . . . . .   LITERALLY '405'
159                 HACFRPCOMPLETE . . .   LITERALLY '413'
159                 HACFRPDEFIRB . . . .   LITERALLY '433'
159                 HACFRPPUTDATAP . . .   LITERALLY '412'
159                 HACFRPPUTNEWBLK. . .   LITERALLY '409'
159                 HACFRPPUTPTR . . . .   LITERALLY '411'
159                 HACFRPPUTSENDRBSBACK   LITERALLY '410'
159                 HACFSENDBLKLEN . . .   LITERALLY '403'
159                 HACFTPGETBLKLEN. . .   LITERALLY '408'
159                 HACFTPGETCHK . . . .   LITERALLY '407'
159                 HACFTPIRBACB . . . .   LITERALLY '415'
159                 HACFTPIRBP . . . . .   LITERALLY '414'
159                 HACFTPIRBTYPE. . . .   LITERALLY '406'
159                 HACFTPTIMEOUT. . . .   LITERALLY '435'
159                 HACFTQSEQ. . . . . .   LITERALLY '417'
```

```
159                 HACFXQ . . . . . .   LITERALLY '436'
159                 HACFZQ . . . . . .   LITERALLY '437'
                    HIGH . . . . . .     BUILTIN          359  420  455  506
 48  0000H      4   HOSTP. . . . . .     POINTER IN PROC (DLSOURCE_EQ_HOST) PARAMETER          48
 12                  ILLEGALREQ . . . . .  LITERALLY '10'
368  0624H     47   INCREASERET. . . . .  PROCEDURE IN PROC (UPDATERETRANTIMEOUT) STACK=0006H       384  395
 15  0000H      1   INDEX. . . . . .     BYTE IN PROC (SETUPCDB) PARAMETER          15
 73  0000H      2   INIT . . . . . .     WORD IN PROC (CQCREATESEMAPHORE) PARAMETER      73
240  0004H      1   INSERTCTLBITSFLAG. . .  BYTE IN PROC (SENDRBSBACK) PARAMETER AUTOMATIC       240  242
 12                  INVALIDPOINTER . . . .  LITERALLY '20'
 12                  INVALIDREQ . . . . . .  LITERALLY '2'
  4  0000H      2   IPINMAX. . . . . .   WORD EXTERNAL(16)
  9  0000H      8   IRB. . . . . .       STRUCTURE BASED(IRB0)
     0000H      4       CMXPTR . . . . . .  POINTER
     0004H      1       TYPE . . . . . .    BYTE
     0005H      1       CDBINDEX . . . . .  BYTE
     0006H      2       CID. . . . . .      WORD
 10                  IRBAYTTIMER. . . . . .  LITERALLY '7'
 10                  IRBCTLACBMASK. . . . .  LITERALLY '40H'        454  505
 10                  IRBCTLTIMEOUTMASK. . .  LITERALLY '0C0H'
 54  0000H      2   IRBINDEX0. . . . . .  WORD IN PROC (DEFERIRBTP) PARAMETER       54
 51  0000H      2   IRBINDEX0. . . . . .  WORD IN PROC (SENDDEFERREDIRBS) PARAMETER       51
 10                  IRBINVALIDMASK . . . .  LITERALLY '38H'
 54  0000H      2   IRBLIST0 . . . . . .  WORD IN PROC (DEFERIRBTP) PARAMETER       54
 51  0000H      2   IRBLIST0 . . . . . .  WORD IN PROC (SENDDEFERREDIRBS) PARAMETER       51
 10                  IRBMAXCODE . . . . . .  LITERALLY '7'
 10                  IRBNULL. . . . . .   LITERALLY '0FFH'       195  517
  9  002EH      2   IRB0 . . . . . .     WORD AT
  9  002EH      4   IRBP . . . . . .     POINTER          9
 10                  IRBSENDCHECK . . . . .  LITERALLY '4'          419  426
 10                  IRBSENDFIN . . . . . .  LITERALLY '2'
 10                  IRBSENDFLAG. . . . . .  LITERALLY '5'          291  319  484  494  497  531  543
 10                  IRBSENDRST . . . . . .  LITERALLY '3'          207
 10                  IRBSENDSYN . . . . . .  LITERALLY '0'          505
 10                  IRBSENDSYNACK. . . . .  LITERALLY '1'          474
 10                  IRBTIMEOUTMASK . . . .  LITERALLY '80H'        419  454  505
 10                  IRBTIMEWAITTO. . . . .  LITERALLY '6'          454
 10                  IRBTYPEMASK. . . . . .  LITERALLY '0FH'
  3  0000H          LCIDVECTOR . . . . . .  WORD ARRAY(0) EXTERNAL(0)          469  718  730
  9  0000H     24   LIRB . . . . . .     STRUCTURE BASED(LIRB0)
     0000H      4       CMXPTR . . . . . .  POINTER
     0004H      1       TYPE . . . . . .    BYTE        207*
     0005H      1       REASON . . . . . .  BYTE        208*
     0006H      6       DLDEST . . . . . .  WORD ARRAY(3)          209*  210*  211*
     000CH      2       DESTPORT . . . . .  WORD        212*
     000EH      2       SOURCEPORT . . . .  WORD        213*
     0010H      2       DESTCID. . . . . .  WORD        214*
     0012H      2       SOURCECID. . . . .  WORD        215*
     0014H      2       SEGSEQNO . . . . .  WORD        216*
     0016H      2       SEGACKNO . . . . .  WORD        217*
  9  0032H      2   LIRB0. . . . . .     WORD AT
  9  0032H      4   LIRBP. . . . . .     POINTER          9  205*  218
  8                  LISTEN . . . . . .   LITERALLY '0'          470  578
 62  0000H      4   LISTP. . . . . .     POINTER IN PROC (CQCREATELIST) PARAMETER       62
 12                  LOCABORT . . . . . .  LITERALLY '12'
  3  0000H      6   LOCHOST. . . . . .   WORD ARRAY(3) EXTERNAL(5)          651
```

```
  3  0000H    2  LOCNET . . . . . .   WORD EXTERNAL(4)       185
 12             LOCTIMEOUT . . . . .  LITERALLY '16'
  2             LOGRBMIPPORT . . . .  LITERALLY '5'
                LOW. . . . . . . . .  BUILTIN          420  455  506
 45  0000H    2  M. . . . . . . . . .  WORD IN PROC (MIN) PARAMETER        45
 42  0000H    2  M. . . . . . . . . .  WORD IN PROC (MAXMOD64K) PARAMETER      42
 39  0000H    2  M. . . . . . . . . .  WORD IN PROC (GEMOD64K) PARAMETER       39
 36  0000H    2  M. . . . . . . . . .  WORD IN PROC (GTMOD64K) PARAMETER       36
118  0000H    2  MAILBOX0 . . . . . .  WORD IN PROC (CQSETALARM) PARAMETER    118
109  0000H    2  MAILBOX0 . . . . . .  WORD IN PROC (CQICRECEIVE) PARAMETER  109
106  0000H    2  MAILBOX0 . . . . . .  WORD IN PROC (CQISEND) PARAMETER      106
 94  0000H    2  MAILBOX0 . . . . . .  WORD IN PROC (CQCRECEIVE) PARAMETER     94
 91  0000H    2  MAILBOX0 . . . . . .  WORD IN PROC (CQRECEIVE) PARAMETER     91
 88  0000H    2  MAILBOX0 . . . . . .  WORD IN PROC (CQSEND) PARAMETER       88
 85  0000H    2  MAILBOX0 . . . . . .  WORD IN PROC (CQCREATEMAILBOX) PARAMETER      85
  3  0061H    1  MATCHCASE. . . . . .  BYTE       728*  728  730  733  737
  3  0060H    1  MATCHTRIES . . . . .  BYTE
 41  0000H       MAXMOD64K. . . . . .  PROCEDURE WORD EXTERNAL(30) STACK=0000H      415
  2             MAXSEGDATALENLIT . . .  LITERALLY '1480'
  2             MAXSENDSEG . . . . . .  LITERALLY '07H'
148  0000H    2  MBX0 . . . . . . . .  WORD IN PROC (CQMIPCONNECT) PARAMETER      148
142  0000H    2  MBX0 . . . . . . . .  WORD IN PROC (CQDLLCONNECT) PARAMETER      142
106  0000H    4  MESSAGEP . . . . . .  POINTER IN PROC (CQISEND) PARAMETER    106
 88  0000H    4  MESSAGEP . . . . . .  POINTER IN PROC (CQSEND) PARAMETER     88
 44  0000H       MIN. . . . . . . . .  PROCEDURE WORD EXTERNAL(31) STACK=0000H      303
  2             MINPKTLEN. . . . . .  LITERALLY '46'
  3  0000H    4  MINRETRANTIME. . . . .  DWORD EXTERNAL(7)      373  375
  2             MIPECHOPORT. . . . . .  LITERALLY '7'
154  0000H    4  MIP_FORM . . . . . .  POINTER IN PROC (CQMIPGETADDRESS) PARAMETER      154
139  0000H    2  MODIFIER . . . . . .  WORD IN PROC (CQDLLREADC) PARAMETER    139
136  0000H    2  MODIFIER . . . . . .  WORD IN PROC (CQDLLREAD) PARAMETER     136
                MOVW . . . . . . . .  BUILTIN          186  306  688
145  0000H    4  MSGP . . . . . . . .  POINTER IN PROC (CQMIPSEND) PARAMETER      145
 45  0000H    2  N. . . . . . . . . .  WORD IN PROC (MIN) PARAMETER        45
 42  0000H    2  N. . . . . . . . . .  WORD IN PROC (MAXMOD64K) PARAMETER      42
 39  0000H    2  N. . . . . . . . . .  WORD IN PROC (GEMOD64K) PARAMETER       39
 36  0000H    2  N. . . . . . . . . .  WORD IN PROC (GTMOD64K) PARAMETER       36
 12             NORESOURCESRESP. . . .  LITERALLY '4'
  3  0000H    1  NUMCOBS. . . . . . .  BYTE EXTERNAL(2)
139  0000H    2  OBJECT . . . . . . .  WORD IN PROC (CQDLLREADC) PARAMETER    139
136  0000H    2  OBJECT . . . . . . .  WORD IN PROC (CQDLLREAD) PARAMETER     136
 12             OKCLOSEDRESP . . . . .  LITERALLY '9'          643
 12             OKEOMRESP. . . . . . .  LITERALLY '3'          245
 12             OKFINRESP. . . . . . .  LITERALLY '5'          247
 12             OKRESP . . . . . . . .  LITERALLY '1'          241  341  439
402  0052H    2  OLDHISACKNO. . . . . .  WORD IN PROC (RPERRFLOW)     407*  425
402  0054H    2  OLDHISCREDIT . . . . .  WORD IN PROC (RPERRFLOW)     408*  425
  2             ONBOTCLECHOPORT. . . .  LITERALLY '7'
 12             OPENAREQ . . . . . .  LITERALLY '0'
 12             OPENCONFLICT . . . . .  LITERALLY '18'
 12             OPENPREQ . . . . . .  LITERALLY '1'
 70  0000H    2  PCB0 . . . . . . . .  WORD IN PROC (CQCREATEPROCESS) PARAMETER      70
265  0004H    1  PKTEMPTY . . . . . .  BYTE IN PROC (GETNEXTBLK) PARAMETER AUTOMATIC      265  268
128  0000H    2  PKT0 . . . . . . . .  WORD IN PROC (CQDLLTXSEND) PARAMETER    128
148  0000H    1  PORTID . . . . . . .  BYTE IN PROC (CQMIPCONNECT) PARAMETER      148
 12             POSTRBUFREQ. . . . . .  LITERALLY '7'
```

```
 70  0000H    2  PRI. . . . . .    WORD IN PROC (CQCREATEPROCESS) PARAMETER        70
151  0000H    2  PROCEDUREQ . . .  WORD IN PROC (CQMIPREGISTER) PARAMETER       151
446  0815H  815  PROCESSSEGMENT . .  PROCEDURE STACK=0020H       664  672  679  709  723
157  0000H    4  PTR. . . . . .    POINTER IN PROC (CQMIPGETMIPFORM) PARAMETER          157
228  018AH  441  PUT. . . . . .    PROCEDURE STACK=0018H       430
229  0000H   32  PUTRBS . . . . .  STRUCTURE BASED(PUTRBSP) IN PROC (PUT)
     0000H    1      CONTENTS . . .  BYTE
     0001H    1      CREDIT . . . .  BYTE       251
     0002H    2      LASTSEQ. . . .  WORD
     0004H    4      MIPBUFBASE . .  POINTER
     0008H    2      MIPLENGTH. . .  WORD
     000AH    1      MIPIDSID . . .  BYTE
     000BH    1      MIPOWNERDEVID. .  BYTE
     000CH    2      INTERNALPROCESSID.  WORD
     000EH    1      REQ. . . . . .  BYTE
     000FH    1      RESP . . . . .  BYTE       241*  245*  247*  247
     0010H    2      RTNMIPSKT. . .  WORD
     0012H    4      LINK . . . . .  POINTER          249  250*
     0016H    2      CID. . . . . .  WORD
     0018H    2      FIRSTSEQ . . .  WORD
     001AH    2      CLIENTUSE. . .  WORD
     001CH    2      BUFLEN . . . .  WORD       311*  311
     001EH    1      NUMBLKS. . . .  BYTE       231  267
     001FH    1      VB . . . . . .  BYTE       258  300
229  0046H    4  PUTRBSP. . . . . .  POINTER IN PROC (PUT)       231  247  249  251  252  257*  258
                                      267  299*  300  311
229  0000H    6  PUTRBV . . . . . .  STRUCTURE BASED(PUTRBVP) ARRAY(1) IN PROC (PUT)
     0000H    4      BLKPTR . . . .  POINTER          235  301
     0004H    2      BLKLEN . . . .  WORD       234
229  004AH    4  PUTRBVP. . . . . .  POINTER IN PROC (PUT)       234  235  258*  300*  301
 11  0000H   32  RBS. . . . . .    STRUCTURE BASED(RBSP)
     0000H    1      CONTENTS . . .  BYTE
     0001H    1      CREDIT . . . .  BYTE
     0002H    2      LASTSEQ. . . .  WORD
     0004H    4      MIPBUFBASE . .  POINTER
     0008H    2      MIPLENGTH. . .  WORD
     000AH    1      MIPIDSID . . .  BYTE
     000BH    1      MIPOWNERDEVID. .  BYTE
     000CH    2      INTERNALPROCESSID.  WORD
     000EH    1      REQ. . . . . .  BYTE
     000FH    1      RESP . . . . .  BYTE
     0010H    2      RTNMIPSKT. . .  WORD
     0012H    4      LINK . . . . .  POINTER
     0016H    2      CID. . . . . .  WORD
     0018H    2      FIRSTSEQ . . .  WORD
     001AH    2      CLIENTUSE. . .  WORD
     001CH    2      BUFLEN . . . .  WORD
     001EH    1      NUMBLKS. . . .  BYTE
     001FH    1      VB . . . . . .  BYTE
434  0000H   32  RBS. . . . . .    STRUCTURE BASED(RBSP) IN PROC (CHKDEFERREDSTATUS)
     0000H    1      CONTENTS . . .  BYTE
     0001H    1      CREDIT . . . .  BYTE
     0002H    2      LASTSEQ. . . .  WORD
     0004H    4      MIPBUFBASE . .  POINTER
     0008H    2      MIPLENGTH. . .  WORD
     000AH    1      MIPIDSID . . .  BYTE
```

```
       0006H     1       MIPOWNERDEVID. . .    BYTE
       000CH     2       INTERNALPROCESSID.    WORD
       000EH     1       REQ. . . . . .        BYTE
       000FH     1       RESP . . . . . .      BYTE        439*  441*
       0010H     2       RTNMIPSKT. . . . .    WORD
       0012H     4       LINK . . . . . .      POINTER           437  438*
       0016H     2       CID. . . . . .        WORD
       0018H     2       FIRSTSEQ . . . . .    WORD
       001AH     2       CLIENTUSE. . . . .    WORD
       001CH     2       BUFLEN . . . . . .    WORD
       001EH     1       NUMBLKS. . . . . .    BYTE        440
       001FH     1       VB . . . . . .        BYTE
  30   0000H     4   RBSP . . . . . .          POINTER IN PROC (GETSTATUSINFO) PARAMETER       30
  11   0036H     4   RBSP . . . . . .          POINTER
 434   0056H     4   RBSP . . . . . .          POINTER IN PROC (CHKDEFERREDSTATUS)      435*  436  437  440  441
                                                 442  443*
  11   0000H     6   RBV. . . . . .            STRUCTURE BASED(RBVP) ARRAY(1)
       0000H     4       BLKPTR . . . . . .    POINTER
       0004H     2       BLKLEN . . . . . .    WORD
  11   003AH     4   RBVP . . . . . .          POINTER
 684   0C4CH   413   RCVPROC. . . . . .         PROCEDURE PUBLIC STACK=002AH
 199   0004H     1   REASONCODE . . . . . .    BYTE IN PROC (SENDRSTREPLY) PARAMETER AUTOMATIC      199  208
 543   0986H         REJECTPACKET . . . . .    LABEL IN PROC (PROCESSSEGMENT)     536  541
  12               REMABORT . . . . . .      LITERALLY '14'          223  509  591
  12               REQMAX . . . . . .        LITERALLY '8'
 173   0004H     1   RESPCODE . . . . . .      BYTE IN PROC (TRYTODELETECDB) PARAMETER AUTOMATIC          173  176
                                                 178  179
 447   0B44H    68   RESTOFFINWAIT2 . . . .    PROCEDURE IN PROC (PROCESSSEGMENT) STACK=0010H       616  630
   3   0000H     1   RETRANINCREASE . . . .    BYTE EXTERNAL(8)       369
   3   0000H     2   RETRANWEIGHT . . . . .    WORD EXTERNAL(9)
 139   0000H     4   RETURNBUFP . . . . . .    POINTER IN PROC (CQDLLREADC) PARAMETER         139
 136   0000H     4   RETURNBUFP . . . . . .    POINTER IN PROC (CQDLLREAD) PARAMETER          136
   5   0000H    41   RP . . . . . .            STRUCTURE BASED(RPO)
       0000H     4       KAOSMSGHDR . . . .    POINTER
       0004H     2       BUFLEN . . . . . .    WORD
       0006H     6       DLDEST . . . . . .    WORD ARRAY(3)
       000CH     6       DLSOURCE . . . . .    WORD ARRAY(3)          688
       0012H     2       DLTYPE . . . . . .    WORD
       0014H     2       TCLVERSION . . . .    WORD
       0016H     2       DESTPORT . . . . .    WORD
       0018H     2       SOURCEPORT . . . .    WORD
       001AH     2       DESTCID. . . . . .    WORD
       001CH     2       SOURCECID. . . . .    WORD
       001EH     2       SEGSEQNO . . . . .    WORD
       0020H     2       SEGACKNO . . . . .    WORD
       0022H     2       SEGDATALEN . . . .    WORD
       0024H     2       CTL. . . . . .        WORD
       0026H     2       CHECKSUM . . . . .    WORD
       0028H     1       SEGDATA. . . . . .    BYTE ARRAY(1)          306  308
       0000H         RP . . . . . .            PROCEDURE STACK=0000H
 161   0000H    27   RPDEFERIRBTP . . . . .    PROCEDURE STACK=0010H        196  291  319  426  531  543
 401   069FH   258   RPERRFLOW. . . . . .      PROCEDURE STACK=001CH        487  563  601  612  629  636  640
 160   0068H     1   RPIRBINDEX . . . . . .    BYTE INITIAL     163  742
 160   0065H     3   RPIRBLIST. . . . . .      BYTE ARRAY(3) INITIAL         163  742
 229   0044H     2   RPLENLEFT. . . . . .      WORD IN PROC (PUT)     297*  303  312*  312  315  329
   4   0000H     2   RPMBX. . . . . .          WORD EXTERNAL(15)      686
```

```
  5  0026H     2  RPO. . . . . .    WORD AT          306  308  688  691  740
  5  0026H     4  RPP. . . . . .    POINTER            5  686*
  3  0006H     4  RPROUNDTRIP. . .  DWORD          3  358*  359  360*  360  361*  361  365
  3  0008H     2  RPROUNDTRIPHI. .  WORD AT
  3  0006H     2  RPROUNDTRIPLO. .  WORD AT
  3  0002H     4  RPTIMESTAMP. . .  DWORD          356  358
  3  000AH    28  RP_. . . . . .    STRUCTURE PUBLIC
     0000H     2      DLSOURCE0. .  WORD           186  209  688
     0002H     2      DLSOURCE1. .  WORD           210
     0004H     2      DLSOURCE2. .  WORD           211
     0006H     2      DLTYPE . . .  WORD
     0008H     2      TCLVERSION .  WORD           696
     000AH     2      DESTPORT . .  WORD           213  657  708
     000CH     2      SOURCEPORT .  WORD           187  212  660  668  708
     000EH     2      DESTCID. . .  WORD           215  703  705
     0010H     2      SOURCECID. .  WORD           188  214  525  651  721
     0012H     2      SEGSEQNO . .  WORD           190  217  274  290  318  450  515  534  539  566
                                                   604  621
     0014H     2      SEGACKNO . .  WORD           183  216  405  410  412  415  481  515  553  613
                                                   641
     0016H     2      SEGDATALEN .  WORD           280  285  297  429  532
     0018H     2      CTL. . . . .  WORD           189  200  221  244  280  298  409  429  472  480
                                                   482  490  500  530  532  555  689  690  701
     001AH     2      CHECKSUM . .  WORD           691
  3  0062H     1  RP_HAS_ACK . . .  BYTE           403  513  551  613  641  689*
  3  0063H     1  RP_HAS_FIN . . .  BYTE           246  288  448  564  602  619  690*
  6             RSTACKMASK . . .  LITERALLY '1100H'      480
 13             RSTCLIENTABORT .  LITERALLY '6'
 13             RSTCONNCLOSED. .  LITERALLY '2'          647
 13             RSTILLEGALACK. .  LITERALLY '7'          596
  6             RSTMASK. . . . .  LITERALLY '0100H'      200  221  500  555
 13             RSTNOMATCH . . .  LITERALLY '3'          712
 13             RSTOLDDUPL . . .  LITERALLY '1'          520  527  710
 13             RSTSYNREFUSED. .  LITERALLY '5'          738
 13             RSTVERSIONMISMATCH  LITERALLY '8'        698
 13             RSTZERODESTCID .  LITERALLY '4'          714
 21  0000H     1  RTNCODE. . . . .  BYTE IN PROC (CLEARLISTS) PARAMETER       21
 18  0000H     1  RTNCODE. . . . .  BYTE IN PROC (DELETECDB) PARAMETER        18
  4  0000H     2  SCHEDLOCK. . . .  WORD EXTERNAL(20)      204  206  687  741
  3  0064H     1  SCRATCH. . . . .  BYTE       321*  323*  633*
 32  0000H       SEARCH_LCIDVECTOR  PROCEDURE WORD EXTERNAL(27) STACK=0000H       705
 24  0000H     2  SEGO . . . . . .  WORD IN PROC (CHKSUMCALC) PARAMETER       24
656  0004H     1  SELECTCASE . . .  BYTE IN PROC (CONNECTMATCH) PARAMETER AUTOMATIC      656  659
650  0BC6H    37  SELFCONNECT. . .  PROCEDURE BYTE STACK=0008H         662  670  677  721
103  0000H     2  SEMAPHORE0 . . .  WORD IN PROC (CQICWAIT) PARAMETER       103
100  0000H     2  SEMAPHORE0 . . .  WORD IN PROC (CQISIGNAL) PARAMETER      100
 82  0000H     2  SEMAPHORE0 . . .  WORD IN PROC (CQCWAIT) PARAMETER        82
 79  0000H     2  SEMAPHORE0 . . .  WORD IN PROC (CQWAITSEM) PARAMETER       79
 76  0000H     2  SEMAPHORE0 . . .  WORD IN PROC (CQSIGNAL) PARAMETER        76
 73  0000H     2  SEMAPHORE0 . . .  WORD IN PROC (CQCREATESEMAPHORE) PARAMETER     73
 50  0000H       SENDDEFERREDIRBS   PROCEDURE EXTERNAL(33) STACK=0000H       742
 12             SENDEOMREQ . . .  LITERALLY '6'
239  03CCH   171  SENDRBSBACK. . .  PROCEDURE BYTE IN PROC (PUT) STACK=000EH       268  323
 12             SENDREQ. . . . .  LITERALLY '5'
198  0119H   126  SENDRSTREPLY . .  PROCEDURE STACK=000CH        520  527  596  647  698  710  712
                                                   714  738
```

```
  363  0590H    27  SETRETRANTIMEOUT . . .   PROCEDURE STACK=000AH        485  518  557
   14  0000H        SETUPCDB . . . . . .     PROCEDURE EXTERNAL(21) STACK=0000H      707  720  732
  230  0373H    89  SETUPNEWBLK. . . . . .   PROCEDURE IN PROC (PUT) STACK=0008H     259  270
                   SHL. . . . . .           BUILTIN          361  375
                   SHR. . . . . .           BUILTIN          369  372
  145  0000H     2  SOCKET . . . . . .       WORD IN PROC (CQMIPSEND) PARAMETER      145
    3  0000H        SPECTYPE . . . . . .     BYTE ARRAY(0) EXTERNAL(1)          175*  194*  194  576*  576  730
   70  0000H     2  STACK0 . . . . . .       WORD IN PROC (CQCREATEPROCESS) PARAMETER      70
  182  0006H     1  STATECODE. . . . . .     BYTE IN PROC (ACCEPTCONN) PARAMETER AUTOMATIC      182  183  191
    8              STATEMASK. . . . . .     LITERALLY '0FH'        547
   12              STATUSREQ. . . . . .     LITERALLY '3'
   56  0000H        STKY_INCR. . . . . .     PROCEDURE EXTERNAL(35) STACK=0000H      282  283  544  545  693
    6              SYNACKMASK . . . . . .   LITERALLY '1800H'      482
    6              SYNMASK. . . . . .       LITERALLY '0800H'      472  490  701
    8              SYNRCVD. . . . . .       LITERALLY '2'          183  474  497
    8              SYNSENT. . . . . .       LITERALLY '1'          478  517
   33  0000H     2  TARGET . . . . . .       WORD IN PROC (SEARCH_LCIDVECTOR) PARAMETER     33
  336  0004H     2  TARGETSEGNO. . . . . .   WORD IN PROC (COMPLETE) PARAMETER AUTOMATIC      336  340
    2              TCLHEADERLEN . . . . .   LITERALLY '20'
    2              TCLMIPPORT . . . . . .   LITERALLY '4'
    2              TCLPROTOCOLCODE. . . .   LITERALLY '5001H'
    2              TCLPROTOCOLCODEREV . .   LITERALLY '0150H'
    3  0000H     2  TCLVERSION . . . . . .   WORD EXTERNAL(6)       696
    2              TCLVERSIONLIT. . . . .   LITERALLY '101H'
    2              TIMEOUTINCREASESTATE .   LITERALLY '1'          376  381  394
    2              TIMEOUTSTEADYSTATE . .   LITERALLY '0'          386
    8              TIMEWAIT . . . . . .     LITERALLY '6'          452
  336  0000H    32  TOPRBS . . . . . .       STRUCTURE BASED(TOPRBSP) IN PROC (COMPLETE)
       0000H     1      CONTENTS . . . . .   BYTE
       0001H     1      CREDIT . . . . . .   BYTE
       0002H     2      LASTSEQ. . . . . .   WORD       340
       0004H     4      MIPBUFBASE . . . .   POINTER
       0008H     2      MIPLENGTH. . . . .   WORD
       000AH     1      MIPIDSID . . . . .   BYTE
       000BH     1      MIPOWNERDEVID. . .   BYTE
       000CH     2      INTERNALPROCESSID.   WORD
       000EH     1      REQ. . . . . .       BYTE       342
       000FH     1      RESP . . . . . .     BYTE       341*
       0010H     2      RTNMIPSKT. . . . .   WORD
       0012H     4      LINK . . . . . .     POINTER          347  348*
       0016H     2      CID. . . . . .       WORD
       0018H     2      FIRSTSEQ . . . . .   WORD
       001AH     2      CLIENTUSE. . . . .   WORD
       001CH     2      BUFLEN . . . . . .   WORD
       001EH     1      NUMBLKS. . . . . .   BYTE
       001FH     1      VB . . . . . .       BYTE
  336  004EH     4  TOPRBSP. . . . . .       POINTER IN PROC (COMPLETE)        339*  340  342  347  349  352*
    3  0000H     2  TOTPKTSREJ . . . . . .   WORD EXTERNAL(10)      545
    3  0000H     2  TOTPKTSRETRAN. . . . .   WORD EXTERNAL(11)
    3  0000H     2  TOTRCVBUFREJ . . . . .   WORD EXTERNAL(12)      283
    4  0000H     2  TPMBX. . . . . .         WORD EXTERNAL(14)      218  420  455  506
  182  0004H     1  TPREQCODE. . . . . .     BYTE IN PROC (ACCEPTCONN) PARAMETER AUTOMATIC      182  195  196
    2              TRUE . . . . . .         LITERALLY 'OFFH'       226  260  271  302  321  323  652  665
                                             673  680  685
  172  0050H    68  TRYTODELETECDB . . . .   PROCEDURE PUBLIC STACK=0018H      223  509  591  643
  173  0000H   124  TTDC . . . . . .         STRUCTURE BASED(TTDCDBP) IN PROC (TRYTODELETECDB)
```

```
                0000H     1       STATE. . . . . .    BYTE        174*
                0001H     1       OWNERDEVICE. . . .  BYTE
                0002H     2       OWNERPROCESSID . .  WORD
                0004H     2       LOCCID . . . . . .  WORD
                0006H     2       LOCPORT. . . . . .  WORD
                0008H     2       REMNET . . . . . .  WORD
                000AH     6       REMHOST. . . . . .  WORD ARRAY(3)
                0010H     2       REMPORT. . . . . .  WORD
                0012H     2       PERSIST. . . . . .  WORD
                0014H     2       ABORTTOHI. . . . .  WORD
                0016H     2       REMCID . . . . . .  WORD
                0018H     4       RETRANTODW . . . .  DWORD
                001CH     2       RESERVED . . . . .  WORD
                001EH     2       TIMEDSEQNO . . . .  WORD
                0020H     4       SEGTRANSTIMEDW . .  DWORD
                0024H     4       CUMRETRANDW. . . .  DWORD
                0028H     2       PERSISTCNT . . . .  WORD
                002AH     4       CBTQHDR. . . . . .  POINTER
                002EH     4       PCBQHDR. . . . . .  POINTER
                0032H     4       DEFSTATUSP . . . .  POINTER
                0036H     2       MYACKNO. . . . . .  WORD
                0038H     2       SEEN . . . . . .    WORD
                003AH     1       MYCREDIT . . . . .  BYTE
                003BH     1       CURBLKINDEX. . . .  BYTE
                003CH     2       CBDATAINDEX. . . .  WORD
                003EH     2       RCVBYTESCONSUMED .  WORD
                0040H     2       CURBLKLENLEFT. . .  WORD
                0042H     2       HISACKNO . . . . .  WORD
                0044H     2       NEXTTRANSMIT . . .  WORD
                0046H     1       CLOSEDREASON . . .  BYTE        176*
                0047H     1       HISCREDIT. . . . .  BYTE
                0048H     2       HIGHESTSENT. . . .  WORD
                004AH     1       CBTQBUFCNT . . . .  BYTE
                004BH     1       PCBQBUFCNT . . . .  BYTE
                004CH     2       PKTSREJ. . . . . .  WORD
                004EH     2       PKTSRETRAN . . . .  WORD
                0050H     2       NOCONFID . . . . .  WORD
                0052H     2       LASTNOCONFID . . .  WORD
                0054H     1       RETRANSMITSTATE. .  BYTE
                0055H     1       SENDFLAG . . . . .  BYTE
                0056H     2       PENDINGRCVDATA . .  WORD
                0058H     2       RCVBUFREJCNT . . .  WORD
                005AH     2       AYTCOUNT . . . . .  WORD
                005CH     4       DATAALARMCB. . . .  WORD ARRAY(2)
                0060H     1       DATAACBIRBTYPE . .  BYTE
                0061H     1       DATAACBFLAG. . . .  BYTE
                0062H    10       DATAACBREM . . . .  BYTE ARRAY(10)
                006CH     4       CTLALARMCB . . . .  WORD ARRAY(2)
                0070H     1       CTLACBIRBTYPE. . .  BYTE
                0071H     1       CTLACBFLAG . . . .  BYTE
                0072H    10       CTLACBREM. . . . .  BYTE ARRAY(10)
 173    000AH     1   TTDCDBINDEX. . . . . .  BYTE IN PROC (TRYTODELETECDB) PARAMETER AUTOMATIC          173  179
 173    0006H     4   TTDCDBP. . . . . . . .  POINTER IN PROC (TRYTODELETECDB) PARAMETER AUTOMATIC      173  177
                      178  179
  54    0000H     1   TYPE . . . . . . . . .  BYTE IN PROC (DEFERIRBTP) PARAMETER       54
 162    0004H     1   TYPE . . . . . . . . .  BYTE IN PROC (RPDEFERIRBTP) PARAMETER AUTOMATIC      162  163
```

```
  142  0000H      2  TYPE . . . . . .    WORD IN PROC (CQDLLCONNECT) PARAMETER          142
   12                 UNKNOWNCIDRESP . .  LITERALLY '6'
  367  05ABH    121  UPDATERETRANTIMEOUT. .  PROCEDURE STACK=000AH        414
  686  0C4FH        WAITFORSEGMENT . . . .  LABEL IN PROC (RCVPROC)
   97  0000H      2  WCB0 . . . . . .    WORD IN PROC (CQMRECEIVE) PARAMETER      97
   57  0000H      4  WDP. . . . . .      POINTER IN PROC (STKY_INCR) PARAMETER          57


MODULE INFORMATION:

     CODE AREA SIZE     = 0DE9H   3561D
     CONSTANT AREA SIZE = 0000H     0D
     VARIABLE AREA SIZE = 006AH    106D
     MAXIMUM STACK SIZE = 002AH     42D
     2678 LINES READ
     1 PROGRAM WARNING
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```