File created: 19-Jan-93 10:23:19 **{DSK}<python>lde>lispcore>sources>BSP.;3**

changes to: (RECORDS BSPSOC ACKPUP BSPSTREAM)

previous date: 4-Jan-93 17:24:25 {DSK}<python>lde>lispcore>sources>BSP.;2

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS


```
(RPAQQ BSPCOMS
       ((DECLARE%: EVAL@COMPILE DONTCOPY                          ; This socket record has both RTP and BSP state info
              (RECORDS BSPSOC ACKPUP BSPSTREAM)
              (CONSTANTS * RTPSTATES)
              (CONSTANTS * RTPEVENTS)
              (CONSTANTS (WORDSPERPORT 3))
              (MACROS RTP.OTHERFN BSP.OTHERFN BSP.INPUT.ERROR BSP.OUTPUT.ERROR \BSPINCFILEPTR))
        (COMS                                                     ; User-level RTP socket manipulation
              (FNS OPENRTPSOCKET CLOSERTPSOCKET \INIT.RTPPROCESS))
        (COMS                                                     ; RTP process
              (FNS \RTP.SOCKET.PROCESS \RTP.HANDLE.INPUT \RTP.HANDLE.PUP \RTP.HANDLE.RFC \RTP.CLEANUP
                   \RTP.ACTION \RTP.ERROR \RTP.SHOW.FAILURE \RTP.FILTER \SEND.ABORT \SEND.ANSWERING.RFC
                   \SEND.END \SEND.ENDREPLY \SEND.RFC \FILLRTPPUP \SETRTPPORTS)
              (FNS \BSPINIT \BSPEVENTFN \BSP.CLOSE.OPEN.SOCKETS))
        (COMS                                                     ; Creating BSP stream
              (FNS OPENBSPSTREAM \SMASHBSPSTREAM BSPOUTPUTSTREAM BSPINPUTSTREAM BSPFRNADDRESS CLOSEBSPSTREAM
                   \BSP.FLUSHINPUT BSPOPENP GETBSPUSERINFO SETBSPUSERINFO)
              (FNS CREATEBSPSTREAM ENDBSPSTREAM))
        (COMS                                                     ; BSP stream functions
              (FNS BSPBIN \BSP.GETNEXTBUFFER BSPPEEKBIN BSPREADP BSPEOFP \BSPBACKFILEPTR \BSP.PREPARE.INPUT
                   \BSP.GETFILEPTR \BSP.DECLARE.FILEPTR \BSP.SETFILEPTR \BSP.SKIPBYTES \BSP.CLEANUP.INPUT
                   BSPBOUT \BSP.OTHERBOUT \BSPWRITEBLOCK BSPFORCEOUTPUT \BSP.SENDBUFFER \BSP.PREPARE.OUTPUT
                   BSPGETMARK BSPPUTMARK BSP.PUTINTERRUPT))
        (COMS                                                     ; BSP pup handler
              (FNS \BSP.HANDLE.INPUT \BSP.HANDLE.ACK \BSP.HANDLE.DATA \BSP.HANDLE.ERROR \BSP.HANDLE.INTERRUPT
                   \BSP.HANDLE.INTERRUPTREPLY \SEND.ACK \SEARCH.OUTPUTQ \SETBSPTIMEOUT \TRANSMIT.STRATEGY))
        (COMS                                                     ; BSP utilities
              (FNS \BSP.DEFAULT.ERROR.HANDLER \BSP.TIMERFN \BSP.FLUSH.SOCKET.QUEUES \FILLBSPPUP BSPHELP))
        [COMS                                                     ; debugging
              (FNS PPSOC PPSOC.CURRENT PRINTTIMER PRINTPUPQUEUE BSPPRINTPUP \RTP.INFO.HOOK)
              (DECLARE%: DONTCOPY (ALISTS (PUPPRINTMACROS 8 9 16 17 18 20]
        (INITRECORDS BSPSOC)
        (SYSRECORDS BSPSOC)
        (DECLARE%: DONTEVAL@LOAD DOCOPY (P (\BSPINIT)))
        (COMS                                                     ; Some of these may want to be constants
              (INITVARS (\BSPSOCKETS)
                     (\RFC.TIMEOUT 2000)
                     (\RTP.DALLY.TIMEOUT 5000)
                     (\RTP.DEFAULTTIMEOUT 30000)
                     (\BSP.MAXPUPS 12)
                     (\BSP.IDLETIMEOUT 15000)
                     (\BSP.OUTSTANDINGDATATIMEOUT 250)
                     (\BSP.MAXPUPALLOC 200)
                     (\BSP.ALLOCHYSTERESIS 50)
                     (\BSP.OVERLAP.DATA.WITH.ACK)
                     (\BSP.INITIAL.MAXPUPALLOC 5)
                     (\BSP.INITIAL.ADATATIMEOUT 1000)
                     (\BSP.MIN.ADATA.TIMEOUT 500)
                     (\BSP.MAX.ADATA.TIMEOUT 10000)
                     (\BSP.INACTIVITY.TIMEOUT 120000)
                     (\BSP.NO.INACTIVITY.TIMEOUT T))
              (GLOBALVARS \BSPSOCKETS \RFC.TIMEOUT \RTP.DALLY.TIMEOUT \RTP.DEFAULTTIMEOUT \BSP.MAXPUPS
                     \BSP.IDLETIMEOUT \BSP.OUTSTANDINGDATATIMEOUT \BSP.MAXPUPALLOC \BSP.ALLOCHYSTERESIS
                     \BSP.OVERLAP.DATA.WITH.ACK \BSP.INITIAL.MAXPUPALLOC \BSP.INITIAL.ADATATIMEOUT
                     \BSP.MIN.ADATA.TIMEOUT \BSP.MAX.ADATA.TIMEOUT \BSP.INACTIVITY.TIMEOUT
                     \BSP.NO.INACTIVITY.TIMEOUT))))

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(DATATYPE BSPSOC ((FRNPORT WORD)
                  (FRNSOCKET FIXP)                                ; Net,host,socket of partner
                  (LCLPORT WORD)
                  (LCLSOCKET FIXP)                                ; Net,host,socket of us
                  (RTPSTATE BYTE)                                 ; The current state of the RTP connection, see RTPSTATES
                  (RTPPROCESS POINTER)                            ; Process handle for RTP demon
                  (RTPEVENT POINTER)                              ; Notified when RTPSTATE changes
                  (PUPSOC POINTER)                                ; The packet-level socket used by us
                  (CONNID POINTER)                                ; A large integer, the connection ID
                  (RTPTIMER POINTER)                              ; Timer used for timing out some RTP steps
```

```
                    (RTPTIMEOUT WORD)                                   ; Timeout for current RTP op, or zero if none
                    (BSPINPUTHANDLER POINTER)                           ; Function that is the top-level loop of the watcher process
                    ;; The rest of this structure is dedicated to handling the BSP
                    (BSPINPUTSTREAM POINTER)                            ; Pointer back to STREAM object
                    (BSPTIMER POINTER)                                  ; Timer for BSP use
                    (BSPTIMEOUT WORD)
                    (BSPFAILUREREASON POINTER)                          ; Why connection was broken or not opened
                    (BSPOTHERPUPFN POINTER)                             ; Called on error, interrupt and non-bsp pups
                    (BSPERRORHANDLER POINTER)                           ; Called for bsp errors
                    (BSPIOTIMEOUT POINTER)                              ; if non-zero will cause prepare.output and prepare.input to
                                                                        ; timeout
                    (RCVBYTEID POINTER)                                 ; ID of as far as we have acked
                    (RCVINTERRUPTID POINTER)                            ; ID of next incoming interrupt
                    (BSPINPUTQ POINTER)                                 ; Queue of all pups we have received
                    (%#UNREADPUPS WORD)                                 ; How many pups do we have before first hole in input
                    (XMITBYTEID POINTER)                                ; Id of next outgoing pup
                    (XMITINTERRUPTID POINTER)                           ; id of next outgoing interrupt
                    (LASTACKID POINTER)                                 ; Id of last ack, i.e.  how far our partner has read us
                    (%#UNACKEDPUPS WORD)
                    (%#UNACKEDBYTES WORD)                               ; how many pups/bytes have we sent that haven't been acked
                    (BSPOUTPUTQ POINTER)                                ; Queue of sent but not acked pups
                    (BYTESPERPUP WORD)                                  ; Maximum size we are allowed to grow pups
                    (PUPALLOC WORD)                                     ; Remaining outgoing pup allocation, i.e.  partner's allocation less
                                                                        ; #UNACKEDPUPS
                    (BYTEALLOC WORD)                                    ; Remaining outgoing byte allocation
                    (MAXPUPALLOC WORD)
                    (PUPALLOCCOUNT WORD)
                    (ADATACOUNT WORD)                                   ; incremented once per AData sent
                    (LASTADATATIME POINTER)                             ; Time last ADATA was sent
                    (ADATATIMEOUT WORD)                                 ; Timeout currently in use for AData
                    (INACTIVITYTIMER POINTER)                           ; Time of last incoming pup on this connection
                    (LISTENING FLAG)                                    ; if socket was opened as a server rather than user
                    (INTERRUPTOUT FLAG)                                 ; an unacked interrupt is outstanding
                    (INTERRUPTIN FLAG)                                  ; an interrupt has been received
                    (ACKPENDING FLAG)                                   ; Adata was received, we need to ack
                    (ACKREQUESTED FLAG)                                 ; We have sent an Adata, are waiting for ack
                    (SENTZEROALLOC FLAG)                                ; Need to send gratuitous ack
                    (BSPNOACTIVITY FLAG)                                ; True if BSPINACTIVITYTIMEOUT has passed with no sign of
                                                                        ; life from other side
                    (BSPUSERSTATE POINTER)                              ; For applications use to do as it pleases
                    (NIL WORD)                                          ; No longer used
                    (IOTIMEOUTFN POINTER)                               ; function to be called when prepare.* timeout
                    (BSPWHENCLOSEDFN POINTER)                           ; Called when connection is closed
                    (BSPINPUTEVENT POINTER)
                    (BSPLOCK POINTER)
                    (BSPINITTIMER POINTER)
                    (BSPFAILURESTRING POINTER)
                    (BSPINACTIVITYTIMEOUT POINTER))
       (BLOCKRECORD BSPSOC ((FRNNET BYTE)
                            (FRNHOST BYTE)
                            (FRNSOCKETHI WORD)
                            (FRNSOCKETLO WORD)
                            (LCLNET BYTE)
                            (LCLHOST BYTE)
                            (LCLSOCKETHI WORD)
                            (LCLSOCKETLO WORD)))
       [ACCESSFNS BSPSOC ((FRNPUPADDRESS (CONS (fetch FRNPORT of DATUM)
                                               (fetch FRNSOCKET of DATUM)))
                          (LCLPUPADDRESS (CONS (fetch LCLPORT of DATUM)
                                               (fetch LCLSOCKET of DATUM]
       ;; Note: I assume record pkg does not break up the first six words (the two ports).  I hope I don't have to force it
       RTPTIMER _ (CREATECELL \FIXP)
       BSPTIMER _ (CREATECELL \FIXP)
       INACTIVITYTIMER _ (CREATECELL \FIXP)
       LASTADATATIME _ (CREATECELL \FIXP)
       BSPINPUTQ _ (NCREATE 'SYSQUEUE)
       BSPOUTPUTQ _ (NCREATE 'SYSQUEUE))

(BLOCKRECORD ACKPUP ((ACKBYTESPERPUP WORD)
                     (ACKPUPS WORD)
                     (ACKBYTES WORD))                                   ; body of ACK pup, giving partner's allocation

       )

[ACCESSFNS BSPSTREAM ((BSPSOC (fetch F1 of DATUM)
                              (replace F1 of DATUM with NEWVALUE))      ; BSPSOC object
                      (BSPOUTPUTSTREAM (fetch F2 of DATUM)
                              (replace F2 of DATUM with NEWVALUE))      ; If this stream is the input side, gives output side
                      (BSPCURRENTPUP (fetch F3 of DATUM)
                              (replace F3 of DATUM with NEWVALUE))      ; PUP whose body is the current buffer.  Could be redundant
                      (MARKPENDING (fetch F4 of DATUM)
                              (replace F4 of DATUM with NEWVALUE))      ; On input, true if next byte is a mark
                      (BSPFILEPTRHI (fetch FW6 of DATUM)
                              (replace FW6 of DATUM with NEWVALUE))
                      (BSPFILEPTRLO (fetch FW7 of DATUM)
```

```
                              (replace FW7 of DATUM with NEWVALUE))
                        (BSPFILEPTR (\MAKENUMBER (fetch BSPFILEPTRHI of DATUM)
                                                 (fetch BSPFILEPTRLO of DATUM))
                              (PROGN (replace BSPFILEPTRHI of DATUM with (LRSH NEWVALUE BITSPERWORD))
                                     (replace BSPFILEPTRLO of DATUM with (LOGAND NEWVALUE MAX.SMALL.INTEGER]
)

(/DECLAREDATATYPE 'BSPSOC
       '(WORD FIXP WORD FIXP BYTE POINTER POINTER POINTER POINTER POINTER WORD POINTER POINTER POINTER WORD
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER WORD POINTER POINTER POINTER WORD WORD
              POINTER WORD WORD WORD WORD WORD WORD POINTER WORD POINTER FLAG FLAG FLAG FLAG FLAG FLAG FLAG
              POINTER WORD POINTER POINTER POINTER POINTER POINTER POINTER POINTER)
       ;; ---field descriptor list elided by lister---
       '80)

(RPAQQ RTPSTATES ((\STATE.CLOSED 0)
                  (\STATE.SENTRFC 1)
                  (\STATE.LISTENING 2)
                  (\STATE.OPEN 3)
                  (\STATE.ENDRECEIVED 4)
                  (\STATE.ENDSENT 5)
                  (\STATE.DALLYING 6)
                  (\STATE.ABORTED 7)))

(DECLARE%: EVAL@COMPILE

(RPAQQ \STATE.CLOSED 0)

(RPAQQ \STATE.SENTRFC 1)

(RPAQQ \STATE.LISTENING 2)

(RPAQQ \STATE.OPEN 3)

(RPAQQ \STATE.ENDRECEIVED 4)

(RPAQQ \STATE.ENDSENT 5)

(RPAQQ \STATE.DALLYING 6)

(RPAQQ \STATE.ABORTED 7)

(CONSTANTS (\STATE.CLOSED 0)
       (\STATE.SENTRFC 1)
       (\STATE.LISTENING 2)
       (\STATE.OPEN 3)
       (\STATE.ENDRECEIVED 4)
       (\STATE.ENDSENT 5)
       (\STATE.DALLYING 6)
       (\STATE.ABORTED 7))
)

(RPAQQ RTPEVENTS
       ((\EVENT.OPEN 0)
        (\EVENT.OPENLISTENING 1)
        (\EVENT.OPENIMMEDIATE 2)
        (\EVENT.CLOSE 3)
        (\EVENT.FORCECLOSE 4)
        (\EVENT.RFC 5)
        (\EVENT.ABORT 6)
        (\EVENT.END 7)
        (\EVENT.ENDREPLY 8)
        (\EVENT.TIMEOUT 9)))

(DECLARE%: EVAL@COMPILE

(RPAQQ \EVENT.OPEN 0)

(RPAQQ \EVENT.OPENLISTENING 1)

(RPAQQ \EVENT.OPENIMMEDIATE 2)

(RPAQQ \EVENT.CLOSE 3)

(RPAQQ \EVENT.FORCECLOSE 4)

(RPAQQ \EVENT.RFC 5)

(RPAQQ \EVENT.ABORT 6)

(RPAQQ \EVENT.END 7)

(RPAQQ \EVENT.ENDREPLY 8)

(RPAQQ \EVENT.TIMEOUT 9)
```

```
(CONSTANTS (\EVENT.OPEN 0)
       (\EVENT.OPENLISTENING 1)
       (\EVENT.OPENIMMEDIATE 2)
       (\EVENT.CLOSE 3)
       (\EVENT.FORCECLOSE 4)
       (\EVENT.RFC 5)
       (\EVENT.ABORT 6)
       (\EVENT.END 7)
       (\EVENT.ENDREPLY 8)
       (\EVENT.TIMEOUT 9))
)

(DECLARE%: EVAL@COMPILE

(RPAQQ WORDSPERPORT 3)

(CONSTANTS (WORDSPERPORT 3))
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS RTP.OTHERFN MACRO ((PUP SOCKET)
                             (SELECTQ (fetch OTHERPUPFN of SOCKET)
                                  (RELEASE.PUP (RELEASE.PUP PUP))
                                  (\BSP.PUPHANDLER
                                      (\BSP.PUPHANDLER PUP SOCKET))
                                  (APPLY* (fetch OTHERPUPFN of SOCKET)
                                         PUP SOCKET))))

(PUTPROPS BSP.OTHERFN MACRO [(PUP SOCKET)
                             (SELECTQ (fetch BSPOTHERPUPFN of SOCKET)
                                  (RELEASE.PUP (RELEASE.PUP PUP))
                                  (APPLY* (fetch BSPOTHERPUPFN of SOCKET)
                                         PUP
                                         (fetch BSPINPUTSTREAM of SOCKET])

(PUTPROPS BSP.INPUT.ERROR MACRO (OPENLAMBDA (STREAM ERRCODE)
                                  (APPLY* (fetch BSPERRORHANDLER of (fetch BSPSOC of STREAM))
                                         STREAM ERRCODE)))

(PUTPROPS BSP.OUTPUT.ERROR MACRO (OPENLAMBDA (STREAM ERRCODE)
                                    (APPLY* (fetch BSPERRORHANDLER of (fetch BSPSOC of STREAM))
                                           STREAM ERRCODE)))

(PUTPROPS \BSPINCFILEPTR MACRO [(STREAM N)
                                (PROG (NEWLO)
                                      (replace BSPFILEPTRLO of STREAM
                                         with (COND
                                                ((IGREATERP (SETQ NEWLO (IPLUS (fetch BSPFILEPTRLO of STREAM)
                                                                               N))
                                                        MAX.SMALL.INTEGER)
                                                 (add (fetch BSPFILEPTRHI of STREAM)
                                                      1)
                                                 (SUB1 (IDIFFERENCE NEWLO MAX.SMALL.INTEGER)))
                                                (T NEWLO])
)
)
```

;; User-level RTP socket manipulation

```
(DEFINEQ
```

(OPENRTPSOCKET

```
  [LAMBDA (FRNPORT MODE PUPSOC CONNID TIMEOUT FAILURESTRING)      (* bvm%: " 6-Oct-86 11:42")
```

;;; Open an RTP socket in given MODE, talking to FRNPORT. If mode is or contains USER, we set up a user RTP, sending an RFC to FRNPORT, with ;;; initial connection id CONNID (default is chosen at random). If mode is or contains SERVER, we merely listen for an RFC from somewhere, and ;;; FRNPORT and CONNID are ignored. If MODE is or contains RETURN, we don't wait around, but return immediately; caller is assumed to be ;;; monitoring the state of the connection. In the case where we wait, TIMEOUT is how long we will wait (msecs) before giving up and returning NIL. On ;;; success, we return a new BSPSOC. PUPSOC is a packet-level socket opened for the connection by the caller; if omitted, one is created. If MODE is ;;; NIL, we open a USER connection and wait for it to succeed.

```
    (RESETLST
        [PROG (SOCKET INITSTATE SOCKET#)
              [COND
                 (FRNPORT (SETQ FRNPORT (ETHERPORT FRNPORT T]
              [COND
                 [(NULL PUPSOC)
                  (SETQ SOCKET# (PUPSOCKETNUMBER (SETQ PUPSOC (OPENPUPSOCKET]
                 [(FIXP PUPSOC)
                  (SETQ PUPSOC (OPENPUPSOCKET (SETQ SOCKET# PUPSOC]
                 (T (SETQ SOCKET# (PUPSOCKETNUMBER (\DTEST PUPSOC 'PUPSOCKET]
              (SETQ SOCKET (create BSPSOC
                                   RTPSTATE _ \STATE.CLOSED
                                   CONNID _ (OR CONNID (RAND 0 16384))
                                   BSPINPUTHANDLER _ (FUNCTION \RTP.HANDLE.INPUT)
                                   BSPOTHERPUPFN _ (FUNCTION RELEASE.PUP)
```

```
                              PUPSOC _ PUPSOC
                              LCLPORT _ (\LOCALPUPADDRESS)
                              LCLSOCKET _ SOCKET#
                              BSPFAILURESTRING _ FAILURESTRING))
            (\INIT.RTPPROCESS SOCKET)                          ; set up a process to monitor this socket
            (push \BSPSOCKETS SOCKET)
            [COND
               (FRNPORT (replace FRNPORT of SOCKET with (CAR FRNPORT))
                        (replace FRNSOCKET of SOCKET with (CDR FRNPORT]
            (COND
               ((NOT MODE)
                (SETQQ MODE USER)))
            (OBTAIN.MONITORLOCK (fetch BSPLOCK of SOCKET)
                  NIL T)
            [RESETSAVE (PROGN SOCKET)
                  '(AND RESETSTATE (CLOSERTPSOCKET OLDVALUE 0]
            (COND
               [(EQMEMB 'USER MODE)
                (COND
                   ((NOT FRNPORT)
                    (ERROR "No foreign port specified")))
                (\RTP.ACTION SOCKET \EVENT.OPEN)               ; Open the connection (send RFC)
                (COND
                   ((EQMEMB 'RETURN MODE)
                    (RETURN SOCKET]
               [(EQMEMB 'SERVER MODE)
                (replace LISTENING of SOCKET with T)
                (\RTP.ACTION SOCKET \EVENT.OPENLISTENING)
                (COND
                   ((EQMEMB 'RETURN MODE)
                    (RETURN SOCKET]
               ((EQ MODE 'RETURN)                              ; Caller just wants to create this thing, putting it immediately open
                (\RTP.ACTION SOCKET \EVENT.OPENIMMEDIATE)
                (RETURN SOCKET))
               (T (\ILLEGAL.ARG MODE)))
            (SETQ INITSTATE (fetch RTPSTATE of SOCKET))
            [COND
               ((NEQ TIMEOUT T)
                (replace BSPINITTIMER of SOCKET with (SETUPTIMER (OR TIMEOUT \RTP.DEFAULTTIMEOUT]
            (until (NEQ (fetch RTPSTATE of SOCKET)
                        INITSTATE)
               do (MONITOR.AWAIT.EVENT (fetch BSPLOCK of SOCKET)
                           (fetch RTPEVENT of SOCKET)))        ; Wait for transaction to happen
            (RETURN (COND
                       ((OR (EQ (fetch RTPSTATE of SOCKET)
                                \STATE.OPEN)
                            (EQ (fetch RTPSTATE of SOCKET)
                                \STATE.ENDRECEIVED))           ; Socket has been opened ok
                        SOCKET)
                       (T                                      ; Give up, flush everything
                          (CLOSERTPSOCKET SOCKET 0)
                          (COND
                             (FAILURESTRING (\RTP.SHOW.FAILURE SOCKET NIL "No Response")))
                          (AND (EQ FAILURESTRING 'RETURN)
                               (fetch BSPFAILUREREASON of SOCKET])])
```

(**CLOSERTPSOCKET**

```
  [LAMBDA (SOCKET TIMEOUT DONTSEND)                            (* bvm%: "29-Mar-85 21:23")
```

;;; Close given RTP socket. This sends the normal end sequence if appropriate. TIMEOUT is how long we will wait for the end to complete normally.
;;; Value returned is true if the socket was closed normally, NIL if aborted. In either case, SOCKET goes away

```
    (PROG (SUCCESS)
          (WITH.MONITOR (fetch BSPLOCK of SOCKET)
              (COND
                 ((NEQ TIMEOUT 0)                              ; Is zero to force a bad connection closed immediately
                  (replace BSPINITTIMER of SOCKET with (SETUPTIMER (OR TIMEOUT \RTP.DEFAULTTIMEOUT)))
                  (\RTP.ACTION SOCKET \EVENT.CLOSE)
                  (until (COND
                            ((SETQ SUCCESS (EQ (fetch RTPSTATE of SOCKET)
                                               \STATE.CLOSED))
                             T)
                            ((EQ (fetch RTPSTATE of SOCKET)
                                 \STATE.ABORTED)
                             (\RTP.ACTION SOCKET \EVENT.FORCECLOSE)
                             T))
                     do                                        ; wait for end handshake
                         (MONITOR.AWAIT.EVENT (fetch BSPLOCK of SOCKET)
                                (fetch RTPEVENT of SOCKET)
                                \RTP.DEFAULTTIMEOUT)))
                 (T (\RTP.ACTION SOCKET \EVENT.FORCECLOSE))))
          (DEL.PROCESS (PROG1 (fetch RTPPROCESS of SOCKET)
                              (replace RTPPROCESS of SOCKET with NIL)))
                                                               ; Deleting the process performs any other cleanup needed, such
                                                               ; as flushing the PUPSOCKET underneath
          (RETURN SUCCESS])
```

(\INIT.RTPPROCESS

```
  [LAMBDA (SOCKET)                                                    (* bvm%: "29-Mar-85 21:42")
```

;;; Creates a process to handle RTP connection on this socket

```
    (PROG ((PROC (ADD.PROCESS (LIST (FUNCTION \RTP.SOCKET.PROCESS)
                                    (KWOTE SOCKET))
                              'NAME
                              'RTP
                              'RESTARTABLE
                              'NO))
           NAME)
          (replace RTPPROCESS of SOCKET with PROC)
          [replace RTPEVENT of SOCKET with (CREATE.EVENT (SETQ NAME (PROCESS.NAME PROC]
          (replace BSPLOCK of SOCKET with (CREATE.MONITORLOCK NAME])

)
```

;; RTP process

```
(DEFINEQ
```

(\RTP.SOCKET.PROCESS

```
  [LAMBDA (BSPSOCKET)                                                 (* bvm%: "29-Mar-85 21:43")
    (DECLARE (SPECVARS BSPSOCKET))                                    ; BSPSOCKET is for use by PPSOC in our INFO hook
```

;;; This is the process that monitors the state of the RTP connection on BSPSOCKET. This better get run periodically

```
    (PROG NIL
          (OBTAIN.MONITORLOCK (fetch BSPLOCK of BSPSOCKET)
                NIL T)
          (RESETSAVE NIL (LIST (FUNCTION \RTP.CLEANUP)
                               BSPSOCKET))
          (PROCESSPROP (THIS.PROCESS)
                'INFOHOOK
                (FUNCTION \RTP.INFO.HOOK))
      LP  (SPREADAPPLY* (fetch BSPINPUTHANDLER of BSPSOCKET)
                BSPSOCKET)
          (GO LP])
```

(\RTP.HANDLE.INPUT

```
  [LAMBDA (BSPSOCKET)                                                 (* bvm%: "29-Mar-85 14:19")
```

;;; Top-level of RTP process while connection is being opened

```
    (LET ((PUPSOC (fetch PUPSOC of BSPSOCKET))
          PUP TIMER)
         (COND
            ((SETQ PUP (GETPUP PUPSOC))                               ; play with incoming pup
             (\RTP.HANDLE.PUP PUP BSPSOCKET)
             (BLOCK))
            (T (MONITOR.AWAIT.EVENT (fetch BSPLOCK of BSPSOCKET)
                       (PUPSOCKETEVENT PUPSOC)
                       [SETQ TIMER (COND
                                      ((NEQ (fetch RTPTIMEOUT of BSPSOCKET)
                                            0)
                                       (fetch RTPTIMER of BSPSOCKET]
                       (AND TIMER T))
               (COND
                  ((AND TIMER (TIMEREXPIRED? TIMER))
                   (\RTP.ACTION BSPSOCKET \EVENT.TIMEOUT])
```

(\RTP.HANDLE.PUP

```
  [LAMBDA (PUP BSPSOCKET)                                             (* bvm%: "29-Mar-85 21:31")
```

;;; Handles incoming PUP on an RTP connection

```
    (SELECTC (fetch PUPTYPE of PUP)
         (\PT.RFC (\RTP.HANDLE.RFC BSPSOCKET PUP)
                  (SETQ PUP NIL))
         (\PT.END (COND
                     ((\RTP.FILTER BSPSOCKET PUP T T)
                      (\RTP.ACTION BSPSOCKET \EVENT.END PUP))))
         (\PT.ENDREPLY (COND
                          ((\RTP.FILTER BSPSOCKET PUP T T)
                           (\RTP.ACTION BSPSOCKET \EVENT.ENDREPLY PUP))))
         (\PT.ABORT [COND
                       ((\RTP.FILTER BSPSOCKET PUP T T)
                        (\RTP.ACTION BSPSOCKET \EVENT.ABORT PUP)
                        (\RTP.SHOW.FAILURE BSPSOCKET PUP (CONCAT "[Abort] " (GETPUPSTRING PUP BYTESPERWORD])
         (\PT.ERROR (COND
                       ((AND (EQ (fetch ERRORPUPCODE of PUP)
                                 \PUPE.NOSOCKET)
```

```
                        (\RTP.FILTER BSPSOCKET PUP T NIL))    ; Treat type 2 errors as abort
                    (\RTP.ACTION BSPSOCKET \EVENT.ABORT PUP)
                    (\RTP.SHOW.FAILURE BSPSOCKET PUP "No Such Socket"))))
        (PROGN (BSP.OTHERFN PUP BSPSOCKET)
               (SETQ PUP NIL)))
    (AND PUP (RELEASE.PUP PUP])
```

(\RTP.HANDLE.RFC

```
[LAMBDA (BSPSOCKET PUP)                                    (* bvm%: "29-Mar-85 12:52")
```

;; RFC received. This may be either an initiating RFC (if we are listening) or an answering RFC (if we have sent out an initiating RFC of our own)

```
    (LET ((DATA (fetch PUPCONTENTS of PUP)))
         [COND
            ((EQ (fetch (PORT NET) of DATA)
                 0)                                        ; Sender didn't know its own net number, but we know it now
             (replace (PORT NET) of DATA with (fetch PUPSOURCENET of PUP]
         (COND
            ((SELECTC (fetch RTPSTATE of BSPSOCKET)
                  (\STATE.LISTENING                        ; Accept all but broadcast pups
                       (NEQ (fetch PUPDESTHOST of PUP)
                            0))
                  (\STATE.SENTRFC                          ; Must match the RFC we sent out
                       (\RTP.FILTER BSPSOCKET PUP T T))
                  ((LIST \STATE.OPEN \STATE.ENDSENT)       ; probably a duplicate.  Make sure it matches the connection we
                                                           ; think we have open
                       (AND (\RTP.FILTER BSPSOCKET PUP NIL T)
                            (EQ (fetch (PORT NETHOST) of DATA)
                                (fetch FRNPORT of BSPSOCKET))
                            (EQ (fetch (PORT SOCKETHI) of DATA)
                                (fetch FRNSOCKETHI of BSPSOCKET))
                            (EQ (fetch (PORT SOCKETLO) of DATA)
                                (fetch FRNSOCKETLO of BSPSOCKET))))
                  NIL)
             (\RTP.ACTION BSPSOCKET \EVENT.RFC PUP))
            (T                                             ; Bad RFC.  Send an Abort in reply
               (SWAPPUPPORTS PUP)
               (replace PUPLENGTH of PUP with (IPLUS \PUPOVLEN BYTESPERWORD))
               (\PUTBASE DATA 0 0)
               (PUTPUPSTRING PUP "RFC refused")            ; explanatory string
               (replace TYPEWORD of PUP with \PT.ABORT)
               (replace EPREQUEUE of PUP with 'FREE)
               (SENDPUP (fetch PUPSOC of BSPSOCKET)
                      PUP])
```

(\RTP.CLEANUP

```
[LAMBDA (SOCKET)                                           (* bvm%: "14-JUN-83 14:48")
```

;; Cleanup called when the RTP process on this socket is deleted. CLOSERTPSOCKET may or may not have been called yet, so send an abort if ;; socket isn't closed yet

```
    (SETQ \BSPSOCKETS (DREMOVE SOCKET \BSPSOCKETS))
    (\RTP.ACTION SOCKET \EVENT.FORCECLOSE)
```

;; May have been flushed already if the socket was aborted and then timed out, so call CLOSEPUPSOCKET with NOERRORFLG T

```
    (CLOSEPUPSOCKET (fetch PUPSOC of SOCKET)
           T)
    [PROG ((FN (fetch BSPWHENCLOSEDFN of SOCKET)))
          (AND FN (APPLY* FN (OR (fetch BSPINPUTSTREAM of SOCKET)
                                 SOCKET]
    (\BSP.FLUSH.SOCKET.QUEUES SOCKET)
    (replace BSPUSERSTATE of SOCKET with NIL)              ; Explicitly delete to avoid problem of circular structures not being
                                                           ; collected
    (replace BSPINPUTSTREAM of SOCKET with NIL])
```

(\RTP.ACTION

```
[LAMBDA (SOCKET EVENT PUP)                                 (* bvm%: " 8-Mar-84 17:52")
```

;;; Runs the RTP 'finite state machine' according to EVENT, one of several things one might want to do to an RTP socket, either intentionally or because ;;; of an arrived pup. In the latter case, PUP is also supplied. Performs the indicated event, changing state if appropriate and setting timeouts if ;;; appropriate

```
    (PROG ((STATE (fetch RTPSTATE of SOCKET))
           NEWSTATE TIMEOUT STREAM)
          (SELECTC EVENT
               (\EVENT.OPEN                                ; Normal opening of a user connection.  Send RFC
                           (COND
                              ((NEQ STATE \STATE.CLOSED)
                               (\RTP.ERROR SOCKET EVENT))
                              (T (\SEND.RFC SOCKET)
                                 (SETQ NEWSTATE \STATE.SENTRFC))))
               (\EVENT.OPENLISTENING                       ; Nothing to do, just prepare to listen for an RFC
                    (COND
                       ((NEQ STATE \STATE.CLOSED)
                        (\RTP.ERROR SOCKET EVENT))
                       (T (SETQ NEWSTATE \STATE.LISTENING))))
```

```
(\EVENT.OPENIMMEDIATE                                   ; Assume RFC done, just put in open state
    (COND
       ((NEQ STATE \STATE.CLOSED)
        (\RTP.ERROR SOCKET EVENT))
       (T (SETQ NEWSTATE \STATE.OPEN))))
(\EVENT.CLOSE                                           ; Try to close connection.  Several cases
             (SETQ NEWSTATE (SELECTC STATE
                                  (\STATE.SENTRFC  ; Tried to open the connection, now giving up
                                       (\SEND.ABORT SOCKET)
                                       \STATE.ABORTED)
                                  (\STATE.OPEN     ; Normal case, send an END
                                               (\SEND.END SOCKET)
                                               \STATE.ENDSENT)
                                  (\STATE.ENDRECEIVED
                                                   ; Other guy decided to END, too, so forget what we were trying
                                                   ; to do and just reply to this END
                                       (\SEND.ENDREPLY SOCKET)
                                       \STATE.DALLYING)
                                  STATE)))
(\EVENT.FORCECLOSE                                      ; If open, abort
    (SELECTC STATE
        ((LIST \STATE.SENTRFC \STATE.OPEN \STATE.ENDRECEIVED \STATE.ENDSENT)
            (\SEND.ABORT SOCKET))
        NIL)
    (SETQ NEWSTATE \STATE.ABORTED))
(\EVENT.RFC                                             ; Received an RFC
           (SELECTC STATE
               (\STATE.SENTRFC                          ; This is the answering RFC.  Its body contains the port we
                                                        ; should talk to after this
                   (\BLT (LOCF (fetch FRNPORT of SOCKET))
                         (fetch PUPCONTENTS of PUP)
                         WORDSPERPORT)
                   (SETQ NEWSTATE \STATE.OPEN))
               ((LIST \STATE.LISTENING \STATE.OPEN \STATE.ENDSENT)
                                                        ; we were listening for someone, and this is their opening RFC,
                                                        ; or possibly a duplicate
                   [COND
                      ((fetch LISTENING of SOCKET)
                       (\SEND.ANSWERING.RFC SOCKET PUP)
                       (COND
                          ((EQ STATE \STATE.LISTENING)
                           (SETQ NEWSTATE \STATE.OPEN])
                   (\RTP.ERROR SOCKET EVENT PUP)))
(\EVENT.ABORT                                           ; Received an ABORT pup
             (SELECTC STATE
                 ((LIST \STATE.CLOSED \STATE.LISTENING)
                                                        ; Shouldn't happen
                     (\RTP.ERROR SOCKET EVENT PUP))
                 NIL)
             (SETQ NEWSTATE \STATE.ABORTED))
(\EVENT.END                                             ; Received END
           (SELECTC STATE
               ((LIST \STATE.OPEN \STATE.ENDRECEIVED)
                                                        ; Note that we have received the end, but don't do anything until
                                                        ; our user decides to accept the END
                   (SETQ STREAM (fetch BSPINPUTSTREAM of SOCKET))
                   (SETQ NEWSTATE (COND
                                      ([OR (AND (fetch BSPCURRENTPUP of STREAM)
                                                (ILESSP (fetch COFFSET of STREAM)
                                                        (fetch CBUFSIZE of STREAM)))
                                           (IGREATERP (fetch %#UNREADPUPS of SOCKET)
                                                  (COND
                                                     ((fetch BSPCURRENTPUP of STREAM)
                                                      1)
                                                     (T 0]
                                                        ; There is still input waiting to be read, so can't end just yet
                                       \STATE.ENDRECEIVED)
                                      (T                ; Okay, we're ready to end
                                         (\SEND.ENDREPLY SOCKET)
                                         \STATE.DALLYING))))
               ((LIST \STATE.ENDSENT \STATE.DALLYING)
                                                        ; We've already sent an END, but other guy wants to end.  Obey.
                   (\SEND.ENDREPLY SOCKET)
                   (SETQ NEWSTATE \STATE.DALLYING))
               (\RTP.ERROR SOCKET EVENT PUP)))
(\EVENT.ENDREPLY                                        ; Received ENDREPLY
    (SELECTC STATE
        (\STATE.ENDSENT                                 ; This is the reply to our END.  Echo ENDREPLY so partner can
                                                        ; stop dallying
            (\SEND.ENDREPLY SOCKET)
            (SETQ NEWSTATE \STATE.CLOSED))
        (\STATE.DALLYING                                ; We send ENDREPLY to partner's END.  This is the echoing
                                                        ; ENDREPLY, so everything is cool
            (SETQ NEWSTATE \STATE.CLOSED))
        (\RTP.ERROR SOCKET EVENT PUP)))
(\EVENT.TIMEOUT                                         ; RTPTIMER expired, probably want to retransmit something
                                                        ; Might be nice, perhaps, if we kept copies of these pups that we
                                                        ; might want to retransmit
```

```
                (COND
                   ((EQ STATE \STATE.DALLYING)
                    (SETQ NEWSTATE \STATE.CLOSED))
                   ((AND (fetch BSPINITTIMER of SOCKET)
                         (TIMEREXPIRED? (fetch BSPINITTIMER of SOCKET)))
                    (\SEND.ABORT SOCKET)
                    (SETQ NEWSTATE \STATE.CLOSED)
                    (replace BSPINITTIMER of SOCKET with NIL))
                   (T (SELECTC STATE
                           (\STATE.SENTRFC
                                (\SEND.RFC SOCKET))
                           (\STATE.ENDSENT
                                (\SEND.END SOCKET))
                           NIL))))
            (ERROR "Unknown RTP event" EVENT))
         [COND
            (NEWSTATE (replace RTPSTATE of SOCKET with (SETQ STATE NEWSTATE))
                   (NOTIFY.EVENT (fetch RTPEVENT of SOCKET))
                   (AND (fetch BSPINPUTEVENT of SOCKET)
                        (NOTIFY.EVENT (fetch BSPINPUTEVENT of SOCKET]
         (SELECTC STATE
              ((LIST \STATE.SENTRFC \STATE.ENDSENT \STATE.DALLYING)
                   (SETUPTIMER (SETQ TIMEOUT (COND
                                        ((EQ STATE \STATE.DALLYING)
                                         \RTP.DALLY.TIMEOUT)
                                        (T \RFC.TIMEOUT)))
                          (fetch RTPTIMER of SOCKET))
                   (replace RTPTIMEOUT of SOCKET with TIMEOUT))
              (replace RTPTIMEOUT of SOCKET with 0])
```

(**RTP.ERROR**

```
[LAMBDA (SOCKET EVENT FOREIGNPUP)                                   (* bvm%: " 8-Mar-84 17:52")
   (COND
      (PUPTRACEFLG (PRIN1 "[Unexpected RTP event " PUPTRACEFILE)
             (PRINTCONSTANT EVENT RTPEVENTS PUPTRACEFILE "\EVENT.")
             (PRIN1 " when in state " PUPTRACEFILE)
             (PRINTCONSTANT (fetch RTPSTATE of SOCKET)
                    RTPSTATES PUPTRACEFILE "\STATE.")
             (PRIN1 "]
                    " PUPTRACEFILE])
```

(**RTP.SHOW.FAILURE**

```
[LAMBDA (SOCKET PUP REASON)                                         (* bvm%: "29-Mar-85 21:38")
   (LET ((FAILURESTRING (fetch BSPFAILURESTRING of SOCKET)))
        (COND
           ((NEQ FAILURESTRING T)                                   ; Only if we haven't done this already
            (COND
               ((NEQ FAILURESTRING 'RETURN)                         ; RETURN means caller wants to see it, not user
                [COND
                   (PUP (printout PROMPTWINDOW T "From " (ETHERHOSTNAME (fetch PUPSOURCE of PUP)
                                                                        T)))
                   (T (printout PROMPTWINDOW T (ETHERHOSTNAME (fetch FRNPORT of SOCKET)
                                                              T]
                (PRIN1 ": " PROMPTWINDOW)
                (COND
                   (FAILURESTRING (printout PROMPTWINDOW FAILURESTRING " because: ")))
                (PRIN1 REASON PROMPTWINDOW)))
            (replace BSPFAILURESTRING of SOCKET with T)             ; Don't do this again
            (replace BSPFAILUREREASON of SOCKET with REASON])
```

(**RTP.FILTER**

```
[LAMBDA (SOCKET PUP CHECKFRNPORT CHECKID)                           (* bvm%: "29-Mar-85 21:25")
   ;; True if PUP is a valid RTP pup for this socket, checking frnport and/or id as indicated
   (AND (NEQ (fetch PUPDESTHOST of PUP)
             0)
        [OR (NOT CHECKFRNPORT)
            (PROGN [COND
                      ((EQ (fetch (BSPSOC FRNNET) of SOCKET)
                           0)                                       ; We didn't know the local net when we opened the socket;
                                                                    ; perhaps we do now
                       (replace (BSPSOC FRNNET) of SOCKET with (fetch PUPDESTNET of PUP]
                   (AND (EQ (fetch PUPSOURCE of PUP)
                            (fetch (BSPSOC FRNPORT) of SOCKET))
                        (EQ (fetch PUPSOURCESOCKETHI of PUP)
                            (fetch (BSPSOC FRNSOCKETHI) of SOCKET))
                        (EQ (fetch PUPSOURCESOCKETLO of PUP)
                            (fetch (BSPSOC FRNSOCKETLO) of SOCKET]
        (OR (NOT CHECKID)
            (AND (EQ (fetch PUPIDHI of PUP)
                     (\HINUM (fetch CONNID of SOCKET)))
                 (EQ (fetch PUPIDLO of PUP)
                     (\LONUM (fetch CONNID of SOCKET])
```

(\SEND.ABORT

```
[LAMBDA (SOCKET)                                                    (* bvm%: " 8-Mar-84 17:52")
  (PROG ((PUP (ALLOCATE.PUP)))
        (\FILLRTPPUP SOCKET PUP \PT.ABORT (IPLUS BYTESPERWORD \PUPOVLEN))
                                                                    ; Length counts the abort code word
        (\PUTBASE (fetch PUPCONTENTS of PUP)
               0 0)                                                 ; Abort code
        (PUTPUPSTRING PUP (COND
                             ((EQ (fetch RTPSTATE of SOCKET)
                                  \STATE.SENTRFC)
                              "Connection attempt aborted")
                             (T "Connection aborted")))             ; Explanatory string
        (SENDPUP (fetch PUPSOC of SOCKET)
               PUP])
```

(\SEND.ANSWERING.RFC

```
[LAMBDA (SOCKET IPUP)                                               (* bvm%: " 8-Mar-84 17:52")
```

;;; sends an RFC in response to the RFC in IPUP. The connection port we send is self, since we can only support one connection in this model

```
    (PROG ((OPUP (ALLOCATE.PUP)))
          (COND
             ((EQ (fetch RTPSTATE of SOCKET)
                  \STATE.LISTENING)
              ;; We were waiting for this. If not, this is a duplicate RFC and we just throw it away after retransmitting the answering RFC
              (replace CONNID of SOCKET with (fetch PUPID of IPUP))
              (\BLT (LOCF (fetch FRNPORT of SOCKET))
                    (fetch PUPCONTENTS of IPUP)
                    WORDSPERPORT)                                   ; Set foreign connection port for this connection. Our LCLPORT
                                                                    ; should already be correct

              ))
          (\FILLRTPPUP SOCKET OPUP \PT.RFC (IPLUS (UNFOLD WORDSPERPORT BYTESPERWORD)
                                                  \PUPOVLEN))
          (\BLT (LOCF (fetch DEST of OPUP))
                (LOCF (fetch SOURCE of IPUP))
                WORDSPERPORT)                                       ; Send this pup to the port by which IPUP arrived, not by the RTP
                                                                    ; connection port
          (\BLT (fetch PUPCONTENTS of OPUP)
                (LOCF (fetch LCLPORT of SOCKET))
                WORDSPERPORT)                                       ; Our connection port is self
          (replace EPREQUEUE of OPUP with 'FREE)
          (SENDPUP (fetch PUPSOC of SOCKET)
                 OPUP])
```

(\SEND.END

```
[LAMBDA (SOCKET)                                                    (* bvm%: " 8-FEB-83 18:22")
  (SENDPUP (fetch PUPSOC of SOCKET)
         (\FILLRTPPUP SOCKET NIL \PT.END \PUPOVLEN])
```

(\SEND.ENDREPLY

```
[LAMBDA (SOCKET)                                                    (* bvm%: " 8-FEB-83 18:23")
  (SENDPUP (fetch PUPSOC of SOCKET)
         (\FILLRTPPUP SOCKET NIL \PT.ENDREPLY \PUPOVLEN])
```

(\SEND.RFC

```
[LAMBDA (SOCKET)                                                    (* bvm%: "25-Aug-84 23:08")
```

;;; Sends an initiating RFC on SOCKET

```
    (PROG ((PUP (ALLOCATE.PUP)))
          (replace PUPLENGTH of PUP with (OR (IPLUS (UNFOLD WORDSPERPORT BYTESPERWORD)
                                                    \PUPOVLEN)
                                             \PUPOVLEN))
          (replace PUPTYPE of PUP with \PT.RFC)
          (replace PUPID of PUP with (fetch CONNID of SOCKET))
          (\BLT (LOCF (fetch PUPDEST of PUP))
                (LOCF (fetch FRNPORT of SOCKET))
                (TIMES 2 WORDSPERPORT))
          (replace PUPSOURCE of PUP with 0)
          (if (\ROUTE.PUP PUP)
              then                                                  ; Find out what net it will send on, then make that our local port
                   (replace LCLPORT of SOCKET with (fetch PUPSOURCE of PUP)))
          (\BLT (fetch PUPCONTENTS of PUP)
                (LOCF (fetch LCLPORT of SOCKET))
                WORDSPERPORT)                                       ; Connection port = self
          (SENDPUP (fetch PUPSOC of SOCKET)
                 PUP])
```

(\FILLRTPPUP

```
  [LAMBDA (SOCKET PUP TYPE LENGTH)                                    (* bvm%: " 8-FEB-83 18:21")

;;; Fills in an RTP pup for SOCKET.  TYPE is the pup type, LENGTH its length.  We fill in also the ID (connection ID) and local and foreign ports (from
;;; socket)

    (OR PUP (SETQ PUP (ALLOCATE.PUP)))
    (replace PUPLENGTH of PUP with (OR LENGTH \PUPOVLEN))
    (replace TYPEWORD of PUP with TYPE)                               ; Clears TCONTROL while setting TYPE
    (replace PUPID of PUP with (fetch CONNID of SOCKET))
    (\SETRTPPORTS SOCKET PUP)
    PUP])
```

(\SETRTPPORTS

```
  [LAMBDA (SOCKET PUP)                                                (* bvm%: " 2-NOV-83 14:33")
                                                                      ; Fill in both Frn and lcl ports in one move
    (\BLT (LOCF (fetch DEST of PUP))
          (LOCF (fetch FRNPORT of SOCKET))
          (ITIMES WORDSPERPORT 2])

)

(DEFINEQ
```

(\BSPINIT

```
  [LAMBDA NIL                                                         (* bvm%: "28-Apr-85 14:12")
    ;; Defines the BSP device, so that you can BIN and BOUT on BSP streams
    (DECLARE (GLOBALVARS \BSPFDEV))
    (SETQ \BSPFDEV
     (create FDEV
             DEVICENAME _ (FUNCTION BSP)
             RESETABLE _ NIL
             RANDOMACCESSP _ NIL
             PAGEMAPPED _ NIL
             FDBINABLE _ T
             FDBOUTABLE _ T
             BUFFERED _ T
             CLOSEFILE _ (FUNCTION CLOSEBSPSTREAM)
             DELETEFILE _ (FUNCTION NILL)
             GETFILEINFO _ (FUNCTION NILL)
             OPENFILE _ (FUNCTION NILL)
             READPAGES _ (FUNCTION \IS.NOT.RANDACCESSP)
             SETFILEINFO _ (FUNCTION NILL)
             GENERATEFILES _ (FUNCTION \GENERATENOFILES)
             TRUNCATEFILE _ (FUNCTION NILL)
             WRITEPAGES _ (FUNCTION \IS.NOT.RANDACCESSP)
             GETFILENAME _ (FUNCTION NILL)
             REOPENFILE _ (FUNCTION NILL)
             EVENTFN _ (FUNCTION \BSPEVENTFN)
             DIRECTORYNAMEP _ (FUNCTION NILL)
             HOSTNAMEP _ (FUNCTION NILL)
             BIN _ (FUNCTION \BUFFERED.BIN)
             BOUT _ (FUNCTION \BUFFERED.BOUT)
             READP _ (FUNCTION BSPREADP)
             EOFP _ (FUNCTION BSPEOFP)
             PEEKBIN _ (FUNCTION \BUFFERED.PEEKBIN)
             GETFILEPTR _ (FUNCTION \BSP.GETFILEPTR)
             SETFILEPTR _ (FUNCTION \BSP.SETFILEPTR)
             BACKFILEPTR _ (FUNCTION \BSPBACKFILEPTR)
             BLOCKIN _ (FUNCTION \BUFFERED.BINS)
             BLOCKOUT _ (FUNCTION \BSPWRITEBLOCK)
             GETNEXTBUFFER _ (FUNCTION \BSP.GETNEXTBUFFER)
             FORCEOUTPUT _ (FUNCTION BSPFORCEOUTPUT)
             LASTC _ (FUNCTION \ILLEGAL.DEVICEOP)
             GETEOFPTR _ (FUNCTION \IS.NOT.RANDACCESSP)))
    (\DEFINEDEVICE NIL \BSPFDEV])
```

(\BSPEVENTFN

```
  [LAMBDA (DEV EVENT)                                                 (* bvm%: " 8-Mar-84 17:29")
    (SELECTQ EVENT
         (BEFORELOGOUT (\BSP.CLOSE.OPEN.SOCKETS))
         ((AFTERSYSOUT AFTERMAKESYS AFTERLOGOUT AFTERSAVEVM)
              (\BSP.CLOSE.OPEN.SOCKETS)
              (\REMOVEDEVICE.NAMES DEV))
         NIL])
```

(\BSP.CLOSE.OPEN.SOCKETS

```
  [LAMBDA NIL                                                         (* bvm%: "28-Apr-85 14:37")
    (for SOC in (for S in \BSPSOCKETS when (SELECTC (fetch (BSPSOC RTPSTATE) of S)
                                                   ((LIST \STATE.CLOSED \STATE.LISTENING \STATE.ABORTED)
                                                      NIL)
                                                   T)
                   collect S)
       do (WAKE.PROCESS (fetch RTPPROCESS of SOC))
```

```
        ;; Deadlock avoidance.  If process is suspended after exit, as is the default, wake it up, so as to make sure it is not in a place that is holding
        ;; on to the monitor lock.
        (CLOSERTPSOCKET SOC 0)
        (BLOCK])

)

;; Creating BSP stream

(DEFINEQ

(OPENBSPSTREAM
  [LAMBDA (SOCKET OTHERPUPHANDLER ERRORHANDLER IOTIMEOUT IOTIMEOUTFN WHENCLOSEDFN FAILURESTRING)
                                                                    (* bvm%: " 6-Oct-86 11:46")

;;; SOCKET is either an open RTP socket, or a host specification to which to open an rtp socket.  This procedure fills in the parameters to make it a BSP
;;; stream, or returns the result of the OPENRTPSOCKET on failure.

    (PROG (INSTREAM OUTSTREAM SOCKETPROC)
          [COND
             ((NOT (type? BSPSOC SOCKET))                           ; Interpret it as a port to which to establish a user RTP
                                                                    ; connection
              (SETQ SOCKET (OPENRTPSOCKET SOCKET 'USER NIL NIL NIL FAILURESTRING]
          (if (AND SOCKET (type? BSPSOC SOCKET))
              then                                                  ; Check that socket is good
                   (SELECTC (fetch RTPSTATE of SOCKET)
                        ((LIST \STATE.OPEN \STATE.ENDRECEIVED))
                        (RETURN NIL))
            else                                                    ; return possible error message
                 (RETURN SOCKET))
          [replace RCVBYTEID of SOCKET with (replace RCVINTERRUPTID of SOCKET
                                               with (replace XMITBYTEID of SOCKET
                                                       with (replace XMITINTERRUPTID of SOCKET
                                                               with (replace LASTACKID of SOCKET
                                                                       with (fetch CONNID of SOCKET]
                                                                    ; All ID's start out as the connection ID
          (replace ADATATIMEOUT of SOCKET with \BSP.INITIAL.ADATATIMEOUT)
          (replace MAXPUPALLOC of SOCKET with \BSP.INITIAL.MAXPUPALLOC)
          (\BSP.FLUSH.SOCKET.QUEUES SOCKET)
          [replace BSPINPUTSTREAM of SOCKET with (SETQ INSTREAM (create STREAM
                                                                        DEVICE _ \BSPFDEV
                                                                        ACCESS _ 'INPUT]
          (PROGN (replace STRMBOUTFN of INSTREAM with (FUNCTION \BSP.OTHERBOUT))
                                                                    ; For backward compatibility, have to make lisp think we can print
                                                                    ; on the input side
                 (replace ACCESSBITS of INSTREAM with BothBits))
          [replace BSPOUTPUTSTREAM of INSTREAM with (SETQ OUTSTREAM (create STREAM
                                                                           DEVICE _ \BSPFDEV
                                                                           ACCESS _ 'OUTPUT]
          (replace BSPSOC of INSTREAM with (replace BSPSOC of OUTSTREAM with SOCKET))
          [replace %#UNREADPUPS of SOCKET
             with (replace %#UNACKEDPUPS of SOCKET
                     with (replace %#UNACKEDBYTES of SOCKET
                             with (replace PUPALLOC of SOCKET
                                     with (replace BYTESPERPUP of SOCKET
                                             with (replace BYTEALLOC of SOCKET
                                                     with (replace PUPALLOCCOUNT of SOCKET
                                                             with (replace ADATACOUNT of SOCKET with 0]
          (replace BSPFAILURESTRING of SOCKET with 'RETURN)         ; Connection open now, so don't complain when it closes, just
                                                                    ; note reason
          (SETUPTIMER 1 (fetch BSPTIMER of SOCKET))
          (replace BSPTIMEOUT of SOCKET with 1)                     ; \SETBSPTIMEOUT will soon fix this
          (OR (fetch BSPINACTIVITYTIMEOUT of SOCKET)
              (replace BSPINACTIVITYTIMEOUT of SOCKET with \BSP.INACTIVITY.TIMEOUT))
          (SETUPTIMER (fetch BSPINACTIVITYTIMEOUT of SOCKET)
                 (fetch INACTIVITYTIMER of SOCKET))
          (replace BSPINPUTHANDLER of SOCKET with (FUNCTION \BSP.HANDLE.INPUT))
          (replace BSPOTHERPUPFN of SOCKET with (OR OTHERPUPHANDLER (FUNCTION RELEASE.PUP)))
          (replace BSPERRORHANDLER of SOCKET with (OR ERRORHANDLER (FUNCTION \BSP.DEFAULT.ERROR.HANDLER)))
          (replace BSPIOTIMEOUT of SOCKET with (FIXP IOTIMEOUT))
          (replace IOTIMEOUTFN of SOCKET with IOTIMEOUTFN)
          (replace BSPWHENCLOSEDFN of SOCKET with WHENCLOSEDFN)
          (replace BSPINPUTEVENT of SOCKET with (CREATE.EVENT (CONCAT (PROCESS.NAME (SETQ SOCKETPROC
                                                                                      (fetch RTPPROCESS of SOCKET)))
                                                                      "#INPUT")))
          (BLOCK)                                                   ; Let the socket process run to handle any stuff that's arrived
                                                                    ; since the RTP connection was opened
          (WAKE.PROCESS SOCKETPROC)                                 ; It may be stuck in a long timeout
          (RETURN INSTREAM])

(\SMASHBSPSTREAM
  [LAMBDA (OPENSTREAM OLDSTREAM)                                    (* bvm%: "28-OCT-83 18:50")

;;; Hack for use with FTP error recovery.  Copies info from OPENSTREAM into OLDSTREAM, making OLDSTREAM be the stream that controls this
;;; connection
```

```
    (SETQ OLDSTREAM (\DTEST OLDSTREAM 'STREAM))
    (PROG ([SOCKET (fetch BSPSOC of (SETQ OPENSTREAM (\DTEST OPENSTREAM 'STREAM]
           (OUTSTREAM (fetch BSPOUTPUTSTREAM of OPENSTREAM)))
          (with BSPSTREAM OLDSTREAM                                  ; Smash BSP-specific fields
                (SETQ BSPOUTPUTSTREAM OUTSTREAM)
                (SETQ BSPCURRENTPUP (fetch BSPCURRENTPUP of OPENSTREAM))
                (SETQ MARKPENDING (fetch MARKPENDING of OPENSTREAM)))
          (with STREAM OLDSTREAM (SETQ CBUFSIZE (fetch CBUFSIZE of OPENSTREAM))
                (SETQ CPPTR (fetch CPPTR of OPENSTREAM))
                (SETQ COFFSET (fetch COFFSET of OPENSTREAM))
                (SETQ ACCESS (fetch ACCESS of OPENSTREAM)))
          (UNINTERRUPTABLY
              (replace BSPSOC of OLDSTREAM with (replace BSPSOC of OUTSTREAM with SOCKET))
              (replace BSPINPUTSTREAM of SOCKET with OLDSTREAM))])
```

(BSPOUTPUTSTREAM

```
  [LAMBDA (BSPSTREAM)                                                (* bvm%: "10-MAY-83 18:38")
                                                                     ; Returns the output side of a BSPSTREAM
    (ffetch BSPOUTPUTSTREAM of (\DTEST BSPSTREAM 'STREAM])
```

(BSPINPUTSTREAM

```
  [LAMBDA (STREAM)                                                   (* bvm%: "28-Apr-85 13:56")
    (LET [(SOC (ffetch BSPSOC of (\DTEST STREAM 'STREAM]
         (AND SOC (fetch BSPINPUTSTREAM of SOC])
```

(BSPFRNADDRESS

```
  [LAMBDA (STREAM)                                                   (* bvm%: "28-Apr-85 14:00")
    (LET ((SOC (fetch BSPSOC of STREAM)))
         (AND SOC (fetch FRNPUPADDRESS of SOC])
```

(CLOSEBSPSTREAM

```
  [LAMBDA (STREAM TIMEOUT)                                           (* bvm%: "29-Mar-85 21:21")
    ;; Closes BSP stream.  TIMEOUT is how long to wait for partner to agree.  Returns true if closed amiably, NIL if aborted.  SOCKET is dead
    ;; afterwards in any case
    (PROG [(SOCKET (\DTEST (fetch BSPSOC of STREAM)
                           'BSPSOC]
          (OR (FIXP TIMEOUT)
              (SETQ TIMEOUT \RTP.DEFAULTTIMEOUT))
          (WITH.MONITOR (ffetch BSPLOCK of SOCKET)
              (PROG ((INPUTSTREAM (fetch BSPINPUTSTREAM of SOCKET))
                     TIMER)
                    [COND
                       ((SELECTC (fetch RTPSTATE of SOCKET)
                             ((LIST \STATE.OPEN \STATE.ENDRECEIVED)
                                 T)
                             NIL)
                        (BSPFORCEOUTPUT (fetch BSPOUTPUTSTREAM of INPUTSTREAM))
                                                                     ; Send any waiting output, and wait for all our output to be acked
                        (SETQ TIMER (SETUPTIMER TIMEOUT))
                        (while (OR (NEQ (fetch %#UNACKEDPUPS of SOCKET)
                                        0)
                                   (fetch INTERRUPTOUT of SOCKET))
                           do (\BSP.FLUSHINPUT INPUTSTREAM)          ; Discard input while waiting
                              (COND
                                 ((AND (SELECTC (fetch RTPSTATE of SOCKET)
                                           ((LIST \STATE.OPEN \STATE.ENDRECEIVED)
                                               T)
                                           NIL)
                                       (NOT (TIMEREXPIRED? TIMER)))
                                  (MONITOR.AWAIT.EVENT (ffetch BSPLOCK of SOCKET)
                                         (fetch BSPINPUTEVENT of SOCKET)
                                         TIMER T))
                                 (T                                  ; Timed out or connection went bad
                                    (SETQ TIMEOUT 0)
                                    (RETURN]
                    ;; now close the socket, continuing to flush input while we wait
                    (OR (CLOSERTPSOCKET SOCKET TIMEOUT)
                        (SETQ TIMEOUT 0))))
          (BLOCK)
          (RETURN (NEQ TIMEOUT 0])
```

(\BSP.FLUSHINPUT

```
  [LAMBDA (STREAM)                                                   (* bvm%: " 9-MAY-83 16:07")
                                                                     ; Flushes any BSP input currently waiting
    (while (NULL (\BSP.PREPARE.INPUT STREAM 0)) do                   ; Normal data waiting, flush it
                                                  (\BSP.CLEANUP.INPUT STREAM])
```

(BSPOPENP

```
  [LAMBDA (STREAM TYPE)                                              (* bvm%: "25-Aug-84 22:16")
```

;;; True if STREAM is open for the indicated TYPE of i/o: NIL (either), INPUT, OUTPUT, or BOTH. E.g. STREAM may be open for OUTPUT but not
;;; INPUT if partner has requested an end.

```
    (PROG [(SOCKET (fetch BSPSOC of (\DTEST STREAM 'STREAM]
          (RETURN (AND SOCKET (OR (SELECTC (fetch RTPSTATE of SOCKET)
                                           (\STATE.OPEN T)
                                           (\STATE.ENDRECEIVED
                                                (OR (NULL TYPE)
                                                    (EQ TYPE 'OUTPUT)))
                                           (\STATE.ENDSENT
                                                (OR (NULL TYPE)
                                                    (EQ TYPE 'INPUT)))
                                           NIL)
                                  (AND (EQ TYPE 'INPUT)
                                       (OR (IGREATERP (fetch %#UNREADPUPS of SOCKET)
                                                      0)
                                           (ILESSP (fetch COFFSET of STREAM)
                                                   (fetch CBUFSIZE of STREAM])
```

(GETBSPUSERINFO

```
  [LAMBDA (STREAM)                                                   (* bvm%: "10-MAY-83 17:06")
    (ffetch BSPUSERSTATE of (\DTEST (ffetch BSPSOC of (\DTEST STREAM 'STREAM))
                                    'BSPSOC])
```

(SETBSPUSERINFO

```
  [LAMBDA (STREAM VALUE)                                             (* bvm%: " 9-MAY-83 16:12")
    (freplace BSPUSERSTATE of (ffetch BSPSOC of (\DTEST STREAM 'STREAM)) with VALUE])

)

(DEFINEQ
```

(CREATEBSPSTREAM

```
  [LAMBDA (SOCKET OTHERPUPHANDLER ERRORHANDLER IOTIMEOUT IOTIMEOUTFN WHENCLOSEDFN)
                                                                     (* bvm%: "13-JUN-83 18:21")
    (OPENBSPSTREAM SOCKET OTHERPUPHANDLER ERRORHANDLER IOTIMEOUT IOTIMEOUTFN WHENCLOSEDFN])
```

(ENDBSPSTREAM

```
  [LAMBDA (STREAM TIMEOUT)                                           (* bvm%: "13-JUN-83 18:22")
    (CLOSEBSPSTREAM STREAM TIMEOUT])

)
```

;; BSP stream functions

```
(DEFINEQ
```

(BSPBIN

```
  [LAMBDA (STREAM)                                                   (* bvm%: "11-Jul-84 14:44")
    (\BUFFERED.BIN STREAM])
```

(\BSP.GETNEXTBUFFER

```
  [LAMBDA (STREAM WHATFOR NOERRORFLG)                                (* bvm%: "28-Aug-84 21:31")
```

;;; Generic buffer refiller for BSP streams

```
    (PROG (ERRCODE)
          (RETURN (SELECTQ WHATFOR
                       (READ (COND
                                 ((NULL (SETQ ERRCODE (\BSP.PREPARE.INPUT STREAM)))
                                  T)
                                 ((OR (NEQ ERRCODE 'MARK.ENCOUNTERED)
                                      (NULL NOERRORFLG))
                                  (BSP.INPUT.ERROR STREAM ERRCODE))))
                       (WRITE (SETQ STREAM (OR (ffetch BSPOUTPUTSTREAM of (\DTEST STREAM 'STREAM))
                                               STREAM))          ; In case we were given the input side
                              (COND
                                 ((NULL (SETQ ERRCODE (\BSP.PREPARE.OUTPUT STREAM)))
                                  T)
                                 (T (BSP.OUTPUT.ERROR STREAM ERRCODE)
                                                                 ; If that returned, then client must want no error
                                    (RETFROM (OR (STKPOS '\BUFFERED.BOUT)
                                                 (STKPOS '\BUFFERED.BOUTS)
                                                 (ERROR "Bad state for Bout on BSP stream" STREAM))
                                             NIL T))))
                       (SHOULDNT])
```

(BSPPEEKBIN

```
  [LAMBDA (STREAM NOERRORFLG)                                        (* bvm%: "11-Jul-84 15:04")
    (\BUFFERED.PEEKBIN STREAM NOERRORFLG])
```

(BSPREADP

```
  [LAMBDA (STREAM)                                                    (* bvm%: " 7-Feb-84 15:56")
```

;;; true if there is input (not a mark) waiting on STREAM

```
    (PROG (SOCKET)
          (COND
             ((fetch MARKPENDING of STREAM)
              (RETURN NIL))
             ((AND (fetch BSPCURRENTPUP of STREAM)
                   (ILESSP (fetch COFFSET of STREAM)
                           (fetch CBUFSIZE of STREAM)))
              (RETURN T)))
          (RETURN (COND
                     ((IGREATERP (fetch %#UNREADPUPS of (SETQ SOCKET (fetch BSPSOC of STREAM)))
                             (COND
                                ((fetch BSPCURRENTPUP of STREAM)
                                 1)
                                (T 0)))
                      (SELECTC (fetch PUPTYPE of (\QUEUEHEAD (fetch BSPINPUTQ of SOCKET)))
                           ((LIST \PT.MARK \PT.AMARK)
                               NIL)
                           T])
```

(BSPEOFP

```
  [LAMBDA (STREAM)                                                    (* bvm%: "28-Apr-85 14:08")
```

;;; true if bsp STREAM is at end of file, i.e. is at a mark

```
    (COND
       ([NULL (ffetch BSPOUTPUTSTREAM of (SETQ STREAM (\DTEST STREAM 'STREAM]
                                                                      ; Output file is always at EOF.  Not sure EOFP should be used
                                                                      ; this way
        T)
       ((ffetch MARKPENDING of STREAM)
        T)
       ((AND (ffetch BSPCURRENTPUP of STREAM)
             (ILESSP (ffetch COFFSET of STREAM)
                     (ffetch CBUFSIZE of STREAM)))
        NIL)
       (T (EQ (\BSP.PREPARE.INPUT STREAM)
              'MARK.ENCOUNTERED])
```

(\BSPBACKFILEPTR

```
  [LAMBDA (STREAM)                                                    (* bvm%: " 1-JUN-83 12:22")
    (COND
       ((AND (fetch BSPOUTPUTSTREAM of STREAM)
             (fetch CPPTR of STREAM)
             (IGREATERP (fetch COFFSET of STREAM)
                        0))
        (add (fetch COFFSET of STREAM)
             -1))
       (T (ERROR "Can't back up this BSP Stream" STREAM])
```

(\BSP.PREPARE.INPUT

```
  [LAMBDA (STREAM TIMEOUT)                                            (* bvm%: "29-Mar-85 21:23")
```

;;; Prepares INPUP for SOCKET, waiting at most TIMEOUT if supplied (else BSPIOTIMEOUT if in stream else forever). Returns NIL on success, an
;;; error code on failure.

```
    (WITH.MONITOR (fetch BSPLOCK of (fetch BSPSOC of STREAM))
        [PROG (PUP ERRCODE SOCKET)
          LP  (COND
                 [(NULL (fetch BSPCURRENTPUP of STREAM))
                  (SETQ SOCKET (fetch BSPSOC of STREAM))
                  (OR TIMEOUT (SETQ TIMEOUT (fetch BSPIOTIMEOUT of SOCKET)))
                  (BLOCK)
                  ;; Note: we always yield, even before checking to see if pups are available.  That way a process that is sitting reading from the
                  ;; bytestream at least yields once per pup
                  (COND
                     ((SETQ ERRCODE (bind (TIMER _ (AND TIMEOUT (NEQ TIMEOUT 0)
                                                        (SETUPTIMER TIMEOUT)))
                                       do [COND
                                             ((IGREATERP (fetch %#UNREADPUPS of SOCKET)
                                                         0)
                                              (RETURN))
                                             ((NOT (BSPOPENP STREAM 'INPUT))
                                              (RETURN 'BAD.STATE.FOR.BIN))
                                             ((AND TIMEOUT (OR (EQ TIMEOUT 0)
                                                               (TIMEREXPIRED? TIMER)))
                                              (RETURN (COND
```

```
                                                    ((fetch IOTIMEOUTFN of SOCKET)
                                                     (APPLY* (fetch IOTIMEOUTFN of SOCKET)
                                                            STREAM
                                                            'INPUT))
                                                    (T 'BIN.TIMEOUT]
                                           (MONITOR.AWAIT.EVENT (fetch BSPLOCK of SOCKET)
                                                  (fetch BSPINPUTEVENT of SOCKET)
                                                  TIMER TIMER)))
                        (RETURN ERRCODE)))
                   (replace BSPCURRENTPUP of STREAM with (OR (SETQ PUP (\DEQUEUE (fetch BSPINPUTQ of SOCKET)))
                                                             (SHOULDNT)))
                   (replace COFFSET of STREAM with 0)              ; Set byte pointers for reading bytes from pup
                   (replace MARKPENDING of STREAM with (SELECTC (fetch PUPTYPE of PUP)
                                                               ((LIST \PT.MARK \PT.AMARK)
                                                                   (replace CBUFSIZE of STREAM with 0)
                                                                      ; Inhibit BIN microcode from reading mark
                                                                   T)
                                                               (PROGN (replace CPPTR of STREAM
                                                                         with (fetch PUPCONTENTS of PUP))
                                                                      (replace CBUFSIZE of STREAM
                                                                         with (IDIFFERENCE (fetch PUPLENGTH of PUP)
                                                                                           \PUPOVLEN))
                                                                      NIL]
                      ((AND (IGEQ (fetch COFFSET of STREAM)
                                  (fetch CBUFSIZE of STREAM))
                            (NOT (fetch MARKPENDING of STREAM)))       ; Current pup is exhausted
                       (\BSP.CLEANUP.INPUT STREAM)
                       (GO LP)))
                  (RETURN (AND (fetch MARKPENDING of STREAM)
                               'MARK.ENCOUNTERED])])
```

(\BSP.GETFILEPTR

```
  [LAMBDA (STREAM)                                                  (* bvm%: " 2-NOV-83 14:31")
    (IPLUS (fetch BSPFILEPTR of STREAM)
           (COND
              ((fetch CPPTR of STREAM)
               (fetch COFFSET of STREAM))
              (T 0])
```

(\BSP.DECLARE.FILEPTR

```
  [LAMBDA (STREAM ADR)                                              (* bvm%: "28-Apr-85 14:14")
    (replace BSPFILEPTR of STREAM with ADR])
```

(\BSP.SETFILEPTR

```
  [LAMBDA (STREAM INDX)                                             (* bvm%: "28-Apr-85 14:10")
    (PROG (SKIPBYTES)
          (RETURN (COND
                     ((AND (fetch BSPOUTPUTSTREAM of STREAM)
                           (IGEQ (SETQ SKIPBYTES (IDIFFERENCE INDX (\BSP.GETFILEPTR STREAM)))
                                 0))                                ; Can only move file pointer on input, and then only forward
                      (\BSP.SKIPBYTES STREAM SKIPBYTES))
                     (T (\IS.NOT.RANDACCESSP STREAM])
```

(\BSP.SKIPBYTES

```
  [LAMBDA (STREAM NBYTES)                                           (* bvm%: "28-Aug-84 22:37")
    (PROG (ERRCODE BYTESLEFT)
      LP  [COND
             ((SETQ ERRCODE (\BSP.PREPARE.INPUT STREAM))
              (RETURN (BSP.INPUT.ERROR STREAM ERRCODE]
          (COND
             ([IGREATERP NBYTES (SETQ BYTESLEFT (IDIFFERENCE (fetch CBUFSIZE of STREAM)
                                                             (fetch COFFSET of STREAM]
              (SETQ NBYTES (IDIFFERENCE NBYTES BYTESLEFT))
              (replace COFFSET of STREAM with (fetch CBUFSIZE of STREAM))
              (\BSP.CLEANUP.INPUT STREAM)
              (GO LP))
             (T (add (fetch COFFSET of STREAM)
                     NBYTES])
```

(\BSP.CLEANUP.INPUT

```
  [LAMBDA (STREAM)                                                  (* bvm%: " 2-NOV-83 14:23")
```

;;; Called after last byte has been read from this input pup

```
    (PROG [(PUP (\DTEST (fetch BSPCURRENTPUP of STREAM)
                        'ETHERPACKET))
           (SOCKET (\DTEST (fetch BSPSOC of STREAM)
                           'BSPSOC]
          (\BSPINCFILEPTR STREAM (IDIFFERENCE (fetch PUPLENGTH of PUP)
                                              \PUPOVLEN))
          (RELEASE.PUP PUP)
          (replace BSPCURRENTPUP of STREAM with NIL)
```

```
          (replace CBUFSIZE of STREAM with 0)
          (replace CPPTR of STREAM with NIL)
          (add (fetch %#UNREADPUPS of SOCKET)
               -1)
          (COND
             ((fetch SENTZEROALLOC of SOCKET)                    ; Our last ack said we had no allocation, so send a gratuitous ack
                                                                 ; now to get partner going again
              (\SEND.ACK SOCKET])
```

(BSPBOUT

```
  [LAMBDA (STREAM BYTE)                                          (* bvm%: "11-Jul-84 15:03")
    (\BOUT (OR (ffetch BSPOUTPUTSTREAM of (\DTEST STREAM 'STREAM))
               STREAM)
           BYTE])
```

(\BSP.OTHERBOUT

```
  [LAMBDA (STREAM BYTE)                                          (* bvm%: "11-Jul-84 14:52")
    (\BOUT (OR (ffetch BSPOUTPUTSTREAM of (\DTEST STREAM 'STREAM))
               (LISPERROR "FILE NOT OPEN" STREAM))
           BYTE])
```

(\BSPWRITEBLOCK

```
  [LAMBDA (STREAM BASE OFF NBYTES)                               (* bvm%: "11-Jul-84 14:48")
    (\BUFFERED.BOUTS (OR (ffetch BSPOUTPUTSTREAM of (\DTEST STREAM 'STREAM))
                         STREAM)
           BASE OFF NBYTES])
```

(BSPFORCEOUTPUT

```
  [LAMBDA (STREAM DEMANDINGLY)                                   (* bvm%: "11-Jul-84 15:05")
```

;;; Forces any buffered output to be transmitted now. If DEMANDINGLY is true, sends it as an ADATA

```
    (WITH.MONITOR (fetch BSPLOCK of (\DTEST (ffetch BSPSOC of (SETQ STREAM (OR [ffetch BSPOUTPUTSTREAM
                                                                                  of (SETQ STREAM
                                                                                       (\DTEST STREAM 'STREAM]
                                                                                STREAM)))
                                                'BSPSOC))
        [PROG ((PUP (fetch BSPCURRENTPUP of STREAM)))
              (COND
                 (PUP (\BSP.SENDBUFFER STREAM PUP DEMANDINGLY])])
```

(\BSP.SENDBUFFER

```
  [LAMBDA (STREAM PUP DEMANDINGLY)                               (* bvm%: "11-Jul-84 14:36")
```

;;; Transmits PUP, the current output packet of STREAM, then resets stream to output idle. Must be called while owning the bsp lock for this connection

```
    (PROG ((SOCKET (fetch BSPSOC of STREAM))
           (NBYTES (fetch COFFSET of STREAM)))                   ; number of bytes in this pup. Always greater than zero given the
                                                                 ; way we set things up
          (replace PUPLENGTH of PUP with (IPLUS NBYTES \PUPOVLEN))
          (replace PUPID of PUP with (fetch XMITBYTEID of SOCKET))   ; Give it the latest ID, and advance it
          (\SETRTPPORTS SOCKET PUP)
          (replace AUXWORD of PUP with (fetch ADATACOUNT of SOCKET)) ; Lets us know where this pup falls with respect to ADATA's we
                                                                 ; may send
          (UNINTERRUPTABLY
              (add (fetch XMITBYTEID of SOCKET)
                   NBYTES)                                       ; Note: this is wrong if \OVERFLOW ~= 0
              (\BSPINCFILEPTR STREAM NBYTES)
              (add (fetch %#UNACKEDPUPS of SOCKET)
                   1)
              (add (fetch %#UNACKEDBYTES of SOCKET)
                   NBYTES)
              (add (fetch PUPALLOC of SOCKET)
                   -1)                                           ; Adjust allocation information to account for pup/bytes we are
                                                                 ; sending to partner
              (add (fetch BYTEALLOC of SOCKET)
                   (IMINUS NBYTES))
              (replace BSPCURRENTPUP of STREAM with NIL)
              (replace CBUFMAXSIZE of STREAM with (replace CBUFSIZE of STREAM with 0))
              (replace CPPTR of STREAM with NIL)
              (\TRANSMIT.STRATEGY SOCKET PUP (AND DEMANDINGLY T))
                                                                 ; Maybe make it an ADATA
              (replace EPREQUEUE of PUP with (fetch BSPOUTPUTQ of SOCKET))
                                                                 ; Retain pup for possible retransmission
              (SENDPUP (fetch PUPSOC of SOCKET)
                       PUP))
          (\SETBSPTIMEOUT SOCKET])
```

(\BSP.PREPARE.OUTPUT

```
  [LAMBDA (STREAM TIMEOUT)                                       (* bvm%: "29-Mar-85 21:23")
```

```
;;; Prepares OUTPUP for SOCKET, waiting at most TIMEOUT if supplied (else BSPIOTIMEOUT if in stream else forever).  Returns NIL on success, an
;;; error code on failure.  We only need to wait if allocation is exhausted

    (WITH.MONITOR (fetch BSPLOCK of (fetch BSPSOC of STREAM))
        (PROG (PUP ERRCODE SOCKET)
          LP  (COND
                 [(NULL (SETQ PUP (fetch BSPCURRENTPUP of STREAM)))
                  (SETQ SOCKET (fetch BSPSOC of STREAM))
                  (OR TIMEOUT (SETQ TIMEOUT (fetch BSPIOTIMEOUT of SOCKET)))
                  (COND
                     ((SETQ ERRCODE (bind (TIMER _ (AND TIMEOUT (NEQ TIMEOUT 0)
                                                        (SETUPTIMER TIMEOUT)))
                                       do [COND
                                             ((NOT (BSPOPENP STREAM 'OUTPUT))
                                              (RETURN 'BAD.STATE.FOR.BOUT))
                                             ((AND (IGREATERP (fetch PUPALLOC of SOCKET)
                                                              0)
                                                   (IGREATERP (fetch BYTEALLOC of SOCKET)
                                                              0))    ; Partner is ready for us
                                              (RETURN))
                                             ((AND TIMEOUT (OR (EQ TIMEOUT 0)
                                                               (TIMEREXPIRED? TIMER)))
                                              (RETURN (COND
                                                         ((fetch IOTIMEOUTFN of SOCKET)
                                                          (APPLY* (fetch IOTIMEOUTFN of SOCKET)
                                                                  SOCKET
                                                                  'OUTPUT))
                                                         (T 'BOUT.TIMEOUT]
                                          (MONITOR.AWAIT.EVENT (fetch BSPLOCK of SOCKET)
                                                               (fetch BSPINPUTEVENT of SOCKET)
                                                               TIMER TIMER)))
                      (RETURN ERRCODE)))
                  (replace BSPCURRENTPUP of STREAM with (SETQ PUP (ALLOCATE.PUP)))
                  (replace TYPEWORD of PUP with \PT.DATA)
                  (replace CPPTR of STREAM with (fetch PUPCONTENTS of PUP))
                  (replace COFFSET of STREAM with 0)                ; Set counters according to current socket allocation information
                  (replace CBUFMAXSIZE of STREAM with (IMIN (fetch BYTESPERPUP of SOCKET)
                                                            (fetch BYTEALLOC of SOCKET]
                  ((IGEQ (fetch COFFSET of STREAM)
                         (fetch CBUFMAXSIZE of STREAM))
                   (\BSP.SENDBUFFER STREAM PUP)                     ; Send the full packet we have built
                   (GO LP)))
                (RETURN NIL)))])
```

(BSPGETMARK

```
  [LAMBDA (STREAM)                                                  (* bvm%: "11-Jul-84 16:48")
    (COND
       ((EQ (\BSP.PREPARE.INPUT STREAM)
            'MARK.ENCOUNTERED)
        (replace MARKPENDING of STREAM with NIL)
        (PROG1 (\GETBASEBYTE (fetch PUPCONTENTS of (fetch BSPCURRENTPUP of STREAM))
                             0)
               (\BSP.CLEANUP.INPUT STREAM)))
       (T (BSP.INPUT.ERROR STREAM 'BAD.GETMARK])
```

(BSPPUTMARK

```
  [LAMBDA (STREAM MARKBYTE)                                         (* bvm%: "11-Jul-84 15:02")
    (WITH.MONITOR (fetch BSPLOCK of (\DTEST (ffetch BSPSOC of (SETQ STREAM (OR [ffetch BSPOUTPUTSTREAM
                                                                                    of (SETQ STREAM
                                                                                             (\DTEST STREAM 'STREAM]
                                                                                  STREAM)))
                                            'BSPSOC))
        (PROG ((PUP (fetch BSPCURRENTPUP of STREAM))
               ERRCODE)
              (COND
                 (PUP                                               ; Send anything waiting
                      (\BSP.SENDBUFFER STREAM PUP)))
              [COND
                 ((SETQ ERRCODE (\BSP.PREPARE.OUTPUT STREAM))
                  (RETURN (BSP.OUTPUT.ERROR STREAM ERRCODE]
              (\PUTBASEBYTE (ffetch CPPTR of STREAM)
                     (ffetch COFFSET of STREAM)
                     MARKBYTE)
              (add (ffetch COFFSET of STREAM)
                   1)
              (replace PUPTYPE of (SETQ PUP (fetch BSPCURRENTPUP of STREAM)) with \PT.MARK)
              (\BSP.SENDBUFFER STREAM PUP)
              (RETURN MARKBYTE)))])
```

(BSP.PUTINTERRUPT

```
  [LAMBDA (STREAM CODE STRING TIMEOUT)                              (* bvm%: " 1-JUL-83 12:26")
```

```
;;; Sends an Interrupt on SOCKET with given interrupt code and text.  Since there can only be one unacked interrupt outstanding at once, it may have to
;;; wait.  If TIMEOUT is given, we wait only that long.  Returns true on success.
```

```
    (PROG [(SOCKET (\DTEST (fetch BSPSOC of (\DTEST STREAM 'STREAM))
                           'BSPSOC]
          (RETURN (WITH.MONITOR (fetch BSPLOCK of SOCKET)
                       (bind PUP (TIMER _ (AND TIMEOUT (SETUPTIMER TIMEOUT)))
                          do (COND
                                ((OR (NOT (BSPOPENP STREAM 'OUTPUT))
                                     (AND TIMEOUT (TIMEREXPIRED? TIMER)))
                                 (RETURN))
                                ((NOT (fetch INTERRUPTOUT of SOCKET))
                                                          ; State fine for sending interrupt
                                 (SETQ PUP (ALLOCATE.PUP))
                                 (\FILLBSPPUP SOCKET PUP \PT.INTERRUPT (IPLUS \PUPOVLEN BYTESPERWORD)
                                        (fetch XMITINTERRUPTID of SOCKET)
                                        (fetch BSPOUTPUTQ of SOCKET))
                                 (\PUTBASE (fetch PUPCONTENTS of PUP)
                                        0 CODE)           ; Store error code in first data word
                                 (PUTPUPSTRING PUP STRING) ; Append string
                                 (SENDPUP (fetch PUPSOC of SOCKET)
                                        PUP)              ; save pup until it is acked
                                 (replace INTERRUPTOUT of SOCKET with T)
                                 (\SETBSPTIMEOUT SOCKET)
                                 (RETURN T)))
                             (MONITOR.AWAIT.EVENT (fetch BSPLOCK of SOCKET)
                                    (fetch BSPINPUTEVENT of SOCKET)
                                    TIMER TIMER)))])

)

;; BSP pup handler

(DEFINEQ

(\BSP.HANDLE.INPUT
  [LAMBDA (BSPSOCKET)                                     (* bvm%: "29-Mar-85 21:41")

;;; Top-level of RTP process while BSP connection is active

    (PROG ((PUPSOC (fetch PUPSOC of BSPSOCKET))
           (LOCK (fetch BSPLOCK of BSPSOCKET))
           EVENT PUP TIMER)
          (SETQ EVENT (PUPSOCKETEVENT PUPSOC))
      LP  [COND
             ((SETQ PUP (GETPUP PUPSOC))                  ; play with incoming pup
              (SELECTC (fetch PUPTYPE of PUP)
                  ((LIST \PT.MARK \PT.DATA)               ; Ordinary data
                       (\BSP.HANDLE.DATA PUP BSPSOCKET))
                  ((LIST \PT.AMARK \PT.ADATA)             ; Data that demands an ack
                       (replace ACKPENDING of BSPSOCKET with T)
                       (\BSP.HANDLE.DATA PUP BSPSOCKET))
                  (\PT.ACK (\BSP.HANDLE.ACK PUP BSPSOCKET))
                  (\PT.INTERRUPT
                       (\BSP.HANDLE.INTERRUPT PUP BSPSOCKET))
                  (\PT.INTERRUPTREPLY
                       (\BSP.HANDLE.INTERRUPTREPLY PUP BSPSOCKET))
                  (\PT.ERROR (\BSP.HANDLE.ERROR PUP BSPSOCKET))
                  (\RTP.HANDLE.PUP PUP BSPSOCKET)))
             (T (COND
                   ((fetch ACKPENDING of BSPSOCKET)
                    (\SEND.ACK BSPSOCKET)))
                (MONITOR.AWAIT.EVENT LOCK EVENT [SETQ TIMER (COND
                                                      ((NEQ (fetch RTPTIMEOUT of BSPSOCKET)
                                                            0)
                                                       (fetch RTPTIMER of BSPSOCKET))
                                                      ((NEQ (fetch BSPTIMEOUT of BSPSOCKET)
                                                            0)
                                                       (fetch BSPTIMER of BSPSOCKET]
                       (AND TIMER T))
                (COND
                   [(NEQ (fetch RTPTIMEOUT of BSPSOCKET)
                         0)
                    (COND
                       ((TIMEREXPIRED? (fetch RTPTIMER of BSPSOCKET))
                        (\RTP.ACTION BSPSOCKET \EVENT.TIMEOUT]
                   ((NEQ (fetch BSPTIMEOUT of BSPSOCKET)
                         0)
                    (COND
                       ((TIMEREXPIRED? (fetch BSPTIMER of BSPSOCKET))
                        (\BSP.TIMERFN BSPSOCKET]
          (GO LP])
```

(\BSP.HANDLE.ACK
  [LAMBDA (PUP SOCKET)   (* bvm%: "29-May-85 12:31")

;;; Handle an ACK pup. This is a little messy. The ACK's id tells how far partner has gotten in the stream. Assuming this ack was in response to an
;;; ADATA of ours, we need to retransmit anything that we sent before that ADATA which isn't acknowledged in this ack. Finally, the body of the ack

```
;;; gives us an update of partner's allocation

    (PROG (THISID NEXTPUP OLDPUP ADATACOUNT ACKDATA OUTQUEUE INTERRUPTPUP)
          (COND
             ((OR (NOT (\RTP.FILTER SOCKET PUP T))
                  (ILESSP (SETQ THISID (fetch PUPID of PUP))
                          (fetch LASTACKID of SOCKET)))          ; not for us, or is a duplicate/delayed ack
              (RELEASE.PUP PUP)
              (RETURN)))
          [COND
             ((fetch ACKREQUESTED of SOCKET)
              ;; This is presumably in response to our last ADATA, so notice how long it took.  Update our timeout = 2 * avg round trip delay,
              ;; exponentially aged over the last 8 samples
              (replace ADATATIMEOUT of SOCKET
                 with (LRSH [IPLUS (ITIMES 7 (fetch ADATATIMEOUT of SOCKET))
                                   (IMAX \BSP.MIN.ADATA.TIMEOUT (IMIN \BSP.MAX.ADATA.TIMEOUT
                                                                      (LLSH (CLOCKDIFFERENCE (fetch LASTADATATIME
                                                                                                of SOCKET))
                                                                            1]
                            3]
          (replace LASTACKID of SOCKET with THISID)
          (SETQ OUTQUEUE (fetch BSPOUTPUTQ of SOCKET))          ; Now figure out who is acked and who needs retransmitting
          [COND
             ((fetch INTERRUPTOUT of SOCKET)
              (SETQ INTERRUPTPUP (\SEARCH.OUTPUTQ SOCKET T]
          (UNINTERRUPTABLY
              (SETQ OLDPUP (fetch SYSQUEUEHEAD of OUTQUEUE))    ; Empty out the queue and refill it below
              (replace SYSQUEUEHEAD of OUTQUEUE with (replace SYSQUEUETAIL of OUTQUEUE with NIL)))
          (COND
             (INTERRUPTPUP                                      ; Retransmit interrupts immediately
                    (replace EPREQUEUE of INTERRUPTPUP with OUTQUEUE)
                    (SENDPUP (fetch PUPSOC of SOCKET)
                             INTERRUPTPUP)))
          (COND
             ((fetch ACKREQUESTED of SOCKET)
              (SETQ ADATACOUNT (fetch ADATACOUNT of SOCKET))    ; This lets us know whether a pup was sent before or after last
                                                                ; adata
              (replace ACKREQUESTED of SOCKET with NIL)))
          (while OLDPUP do (SETQ NEXTPUP (fetch QLINK of OLDPUP))
                           (replace QLINK of OLDPUP with NIL)
                           (COND
                              ((EQ (fetch PUPTYPE of OLDPUP)
                                   \PT.INTERRUPT)               ; We retransmitted it above, so we should not be seeing this!
                               (\ENQUEUE OUTQUEUE OLDPUP))
                              ((IGEQ (IDIFFERENCE THISID (fetch PUPID of OLDPUP))
                                     (IDIFFERENCE (fetch PUPLENGTH of OLDPUP)
                                                  \PUPOVLEN))   ; has been acked, release it
                               (add (fetch %#UNACKEDPUPS of SOCKET)
                                    -1)
                               (add (fetch %#UNACKEDBYTES of SOCKET)
                                    (IDIFFERENCE \PUPOVLEN (fetch PUPLENGTH of OLDPUP)))
                               (add (fetch PUPALLOCCOUNT of SOCKET)
                                    1)                          ; one more pup successfully received
                               (RELEASE.PUP OLDPUP))
                              ((AND ADATACOUNT (IGREATERP ADATACOUNT (fetch AUXWORD of OLDPUP)))
                                                                ; This pup was originally sent before our last ADATA, so
                                                                ; retransmit it
                               [\TRANSMIT.STRATEGY SOCKET OLDPUP (COND
                                                                    ([AND (fetch QLINK of OLDPUP)
                                                                          (ILEQ ADATACOUNT
                                                                                (fetch AUXWORD
                                                                                   of (fetch QLINK of OLDPUP]
                                                                     (SETQ ADATACOUNT NIL))
                                                                    (T 'NO]
                                                                ; Maybe make it an ADATA if this is the last thing we're
                                                                ; retransmitting, else make it just DATA
                               (replace EPREQUEUE of OLDPUP with OUTQUEUE)
                               (SENDPUP (fetch PUPSOC of SOCKET)
                                        OLDPUP))
                              (T (\ENQUEUE OUTQUEUE OLDPUP)))
                           (SETQ OLDPUP NEXTPUP))
    ;; Now update allocations
          [COND
             ((IGREATERP (fetch PUPALLOCCOUNT of SOCKET)
                         \BSP.ALLOCHYSTERESIS)                  ; We've been doing okay for a while with no congestion, so
                                                                ; increase our max pup allocation
              (replace PUPALLOCCOUNT of SOCKET with 0)
              (COND
                 ((ILESSP (fetch MAXPUPALLOC of SOCKET)
                          \BSP.MAXPUPALLOC)
                  (add (fetch MAXPUPALLOC of SOCKET)
                       1]
          (SETQ ACKDATA (fetch PUPCONTENTS of PUP))
          (replace BYTESPERPUP of SOCKET with (IMIN (fetch ACKBYTESPERPUP of ACKDATA)
                                                   \MAX.PUPLENGTH))
```

```
        (replace PUPALLOC of SOCKET with (IMAX (IMIN (fetch MAXPUPALLOC of SOCKET)
                                                     (IDIFFERENCE (fetch ACKPUPS of ACKDATA)
                                                            (fetch %#UNACKEDPUPS of SOCKET)))
                                               0))                  ; number of pups we can still send
        (replace BYTEALLOC of SOCKET with (IMAX (IDIFFERENCE (fetch ACKBYTES of ACKDATA)
                                                        (fetch %#UNACKEDBYTES of SOCKET))
                                                0))
        (RELEASE.PUP PUP)
        (NOTIFY.EVENT (fetch BSPINPUTEVENT of SOCKET))          ; Actually, notifying that allocation may have changed
        (\SETBSPTIMEOUT SOCKET)
        (SETUPTIMER (fetch BSPINACTIVITYTIMEOUT of SOCKET)
               (fetch INACTIVITYTIMER of SOCKET])
```

(\BSP.HANDLE.DATA

```
  [LAMBDA (PUP SOCKET)                                          (* bvm%: "29-Mar-85 21:23")
```

;;; Processes BSP data and mark pups. Principal task is to figure out where this PUP goes on our input queue.

```
    (PROG (THISID NEWID PREVPUP NEXTPUP DIF DATALENGTH INQUEUE)
          (COND
             ((OR (NOT (\RTP.FILTER SOCKET PUP T))
                  (EQ (SETQ DATALENGTH (IDIFFERENCE (fetch PUPLENGTH of PUP)
                                                    \PUPOVLEN))
                      0)
                  (PROGN ;; if we have no space for incoming pups.  If our partner is a good guy, she pays attention to our allocation reports and
                         ;; never overwhelms us, so this is mainly a problem if someone screws up
                         NIL))                                  ; Pup not for us or is zero-length, so nothing to do
              (RELEASE.PUP PUP)
              (RETURN)))
          (COND
             ((ILEQ (SETQ NEWID (IPLUS (SETQ THISID (fetch PUPID of PUP))
                                       DATALENGTH))
                    (fetch RCVBYTEID of SOCKET))                ; NEWID is id of next byte after this packet.  If less than
                                                                ; RCVBYTEID, it's a duplicate, so discard
              (RELEASE.PUP PUP)
              (RETURN)))
          [COND
             ([OR [NULL (fetch SYSQUEUEHEAD of (SETQ INQUEUE (fetch BSPINPUTQ of SOCKET]
                  (IGREATERP THISID (fetch PUPID of (fetch SYSQUEUETAIL of INQUEUE]
                                                                ; Checking easy case first: pup goes on end of queue
              (\ENQUEUE INQUEUE PUP))
             (T                                                 ; Pup goes somewhere in middle of q
                (SETQ PREVPUP NIL)
                (SETQ NEXTPUP (fetch SYSQUEUEHEAD of INQUEUE))
                (while (NEQ NEXTPUP NIL) do (COND
                                               ((EQ (SETQ DIF (IDIFFERENCE THISID (fetch PUPID of NEXTPUP)))
                                                    0)          ; Is duplicate of NEXTPUP
                                                (RELEASE.PUP PUP)
                                                (RETURN (SETQ PUP NIL)))
                                               ((ILESSP DIF 0)  ; New pup comes before NEXTPUP
                                                (GO $$OUT)))
                                            (SETQ NEXTPUP (fetch QLINK of (SETQ PREVPUP NEXTPUP)))
                   finally                                      ; Insert PUP between PREVPUP and NEXTPUP
                         (COND
                            ((NULL PREVPUP)
                             (replace SYSQUEUEHEAD of INQUEUE with PUP))
                            (T (replace QLINK of PREVPUP with PUP)))
                         (replace QLINK of PUP with NEXTPUP]
```

;; now see if the new pup fills a hole in front of queue, so we can advance our ID of contiguously read pups

```
          (while (AND PUP (IEQP (fetch RCVBYTEID of SOCKET)
                                (fetch PUPID of PUP)))
             do (add (fetch RCVBYTEID of SOCKET)
                     (IDIFFERENCE (fetch PUPLENGTH of PUP)
                                  \PUPOVLEN))                   ; Advance ID past this pup
                (add (fetch %#UNREADPUPS of SOCKET)
                     1)                                         ; One more pup available for BSPBIN
                (NOTIFY.EVENT (fetch BSPINPUTEVENT of SOCKET))
                (SETQ PUP (fetch QLINK of PUP)))
          (SETUPTIMER (fetch BSPINACTIVITYTIMEOUT of SOCKET)
                 (fetch INACTIVITYTIMER of SOCKET))             ; There was non-trivial activity

      ])
```

(\BSP.HANDLE.ERROR

```
  [LAMBDA (PUP BSPSOCKET)                                       (* bvm%: "28-Apr-85 14:38")
```

;;; Handle ERROR pups. The only error codes BSP is interested in are the ones indicating network congestion.

```
    (SELECTC (fetch ERRORPUPCODE of PUP)
        ((LIST \PUPE.SOCKETFULL \PUPE.GATEWAYFULL)
                                                                ;; Port IQ overflow, gateway OQ overflow--congestion error.  Throttle back output by
                                                                ;; decreasing our max outgoing allocation
```

```
            (COND
               ((IGREATERP (fetch MAXPUPALLOC of BSPSOCKET)
                        1)
                (add (fetch MAXPUPALLOC of BSPSOCKET)
                     -1)))
            (replace PUPALLOCCOUNT of BSPSOCKET with 0)          ; Reset hysteresis counter

            )
         (\PUPE.NOSOCKET
            (\RTP.ACTION BSPSOCKET \EVENT.ABORT PUP))
         NIL)                                                    ; Finally pass all errors on to higher-level proc if any
    (BSP.OTHERFN PUP BSPSOCKET])
```

(\BSP.HANDLE.INTERRUPT

```
  [LAMBDA (PUP SOCKET)                                           (* bvm%: "29-Mar-85 21:24")
```

;;; Handles incoming interrupt. Notes that we have an interrupt, and sends an interrupt reply

```
    (COND
       ((\RTP.FILTER SOCKET PUP T)
        (PROG [(DIF (IDIFFERENCE (fetch RCVINTERRUPTID of SOCKET)
                           (fetch PUPID of PUP]
              [COND
                 ((EQ DIF 0)                                   ; New interrupt.  Note receipt and pass on to higher-level handler
                  (add (fetch RCVINTERRUPTID of SOCKET)
                       1)
                  (replace INTERRUPTIN of SOCKET with T)
                  (BSP.OTHERFN PUP SOCKET))
                 (T                                            ; Duplicate or bad ID, discard
                    (RELEASE.PUP PUP)
                    (COND
                       ((NEQ DIF 1)                            ; Garbage
                        (RETURN]
              [SENDPUP (fetch PUPSOC of SOCKET)
                     (\FILLBSPPUP SOCKET NIL \PT.INTERRUPTREPLY \PUPOVLEN (SUB1 (fetch RCVINTERRUPTID
                                                                                  of SOCKET]
                                                                ; reply to it

       ))
       (T                                                      ; Not for us
          (RELEASE.PUP PUP])
```

(\BSP.HANDLE.INTERRUPTREPLY

```
  [LAMBDA (PUP SOCKET)                                           (* bvm%: " 5-JUN-83 15:49")
```

;;; Handles Interrupt Reply. Assuming this is in response to a (the) interrupt we sent out, we can release our copy of the interrupt pup

```
    [COND
       ((AND (\RTP.FILTER SOCKET PUP T)
             (fetch INTERRUPTOUT of SOCKET)
             (IEQP (fetch PUPID of PUP)
                   (fetch XMITINTERRUPTID of SOCKET)))
        (PROG ((INTPUP (\SEARCH.OUTPUTQ SOCKET T)))
              (COND
                 (INTPUP (add (fetch XMITINTERRUPTID of SOCKET)
                              1)
                         (replace INTERRUPTOUT of SOCKET with NIL)  ; In case BSP.PUTINTERRUPT was waiting on us
                         (NOTIFY.EVENT (fetch BSPINPUTEVENT of SOCKET))
                         (RELEASE.PUP INTPUP))
                 (T ;; Inconsistent state: we have INTERRUPTOUT, but can't find the pup on our retransmit queue.  In bcpl implementation the
                    ;; pup might still be on the transmit queue, but here we know we have sent it.  Change this when low-level pup gets into lisp
                    (BSPHELP "Couldn't find interrupt that elicited this reply"]
    (RELEASE.PUP PUP])
```

(\SEND.ACK

```
  [LAMBDA (SOCKET)                                               (* bvm%: "29-Mar-85 21:24")
```

;;; Send an ACK, telling partner how much of the bytestream we have received, and what our current allocation is

```
    (PROG ((PUP (ALLOCATE.PUP))
           [%#PUPS (IMAX 0 (IDIFFERENCE \BSP.MAXPUPS (fetch %#UNREADPUPS of SOCKET]
           DATA)
          ;; Our current allocation is computed by subtracting from our max allocation anything sitting in the input queue.  Don't want to say the length of the
          ;; whole INPUTQ, since stuff after the hole doesn't really count.  This is all approximate, of course, but is sufficient for decent flow control
          (replace ACKPENDING of SOCKET with NIL)
          (\FILLBSPPUP SOCKET PUP \PT.ACK (IPLUS \PUPOVLEN 6)
                 (fetch RCVBYTEID of SOCKET)
                 'FREE)
          (SETQ DATA (fetch PUPCONTENTS of PUP))
          (replace ACKBYTESPERPUP of DATA with \MAX.PUPLENGTH)      ; We can always receive maximal size pups
          (replace ACKPUPS of DATA with %#PUPS)
          (replace ACKBYTES of DATA with (ITIMES %#PUPS \MAX.PUPLENGTH))
```

```
          (replace SENTZEROALLOC of SOCKET with (EQ %#PUPS 0))      ; we said stop.  This will encourage us to send an ack as soon as
                                                                    ; our allocation improves
          (SENDPUP (fetch PUPSOC of SOCKET)
                 PUP)
```

;; At this point the BCPL implementation flushes the pups we have received but not acked, since they will probably be retransmitted anyway. No
;; real need for us to do that, since we don't have a permanently constrained pup pool

```
       ])
```

(\SEARCH.OUTPUTQ

```
  [LAMBDA (SOCKET LOOKFORINTERRUPT)                                 (* bvm%: " 5-JUN-83 15:30")
```

;;; Searches output queue of SOCKET for an interrupt packet, if LOOKFORINTERRUPT is true, or for the last non-interrupt if false, and returns it or NIL

```
    (bind (PUP _ (fetch SYSQUEUEHEAD of (fetch BSPOUTPUTQ of SOCKET)))
          LASTPUP while PUP do (COND
                                  [LOOKFORINTERRUPT (COND
                                                      ((EQ (fetch PUPTYPE of PUP)
                                                           \PT.INTERRUPT)
                                                       (RETURN (\UNQUEUE (fetch BSPOUTPUTQ of SOCKET)
                                                                         PUP]
                                  ((NEQ (fetch PUPTYPE of PUP)
                                        \PT.INTERRUPT)
                                   (SETQ LASTPUP PUP)))
                                (SETQ PUP (fetch QLINK of PUP))
      finally (RETURN (AND LASTPUP (\UNQUEUE (fetch BSPOUTPUTQ of SOCKET)
                                             LASTPUP])
```

(\SETBSPTIMEOUT

```
  [LAMBDA (SOCKET)                                                  (* bvm%: "10-MAY-83 23:11")
```

;;; Sets timer for this socket to wake us up after a while if nothing happens. If we have unacked data outstanding, make this shorter than if we are idle

```
    (SETUPTIMER (replace BSPTIMEOUT of SOCKET with (COND
                                                     [(OR (fetch INTERRUPTOUT of SOCKET)
                                                          (IGREATERP (fetch %#UNACKEDPUPS of SOCKET)
                                                                     0)
                                                          (ILEQ (fetch PUPALLOC of SOCKET)
                                                                0)
                                                          (ILEQ (fetch BYTEALLOC of SOCKET)
                                                                0)
                                                          (ILEQ (fetch BYTESPERPUP of SOCKET)
                                                                0))
                                                                    ; We're waiting for a response
                                                      (WAKE.PROCESS (fetch RTPPROCESS of SOCKET))
                                                                    ; Because we may have shortened the timeout
                                                      (COND
                                                         ((fetch ACKREQUESTED of SOCKET)
                                                                    ; Sent Adata, here's how long we expect to need
                                                          (fetch ADATATIMEOUT of SOCKET))
                                                         (T (IMAX (fetch ADATATIMEOUT of SOCKET)
                                                                  \BSP.OUTSTANDINGDATATIMEOUT]
                                                     (T \BSP.IDLETIMEOUT)))
          (fetch BSPTIMER of SOCKET])
```

(\TRANSMIT.STRATEGY

```
  [LAMBDA (SOCKET PUP MAKEA?)                                       (* bvm%: " 3-MAY-83 11:32")
```

;;; Decides whether to make PUP an ADATA (AMARK) or just DATA (MARK) when MAKEA? is nil. If T it always makes ADATA, if NO it never does.
;;; Current strategy (from BCPL) : demand ack if allocation falls below 1/3 of that given in the last received ack, i.e. if PUPALLOC le
;;; (PUPALLOC+UNACKEDPUPS) /3, or equivalently PUPALLOC*2 le UNACKEDPUPS. If \BSP.OVERLAP.DATA.WITH.ACK is false, however, only
;;; demands ack when allocation is exhausted

```
    (COND
       ([OR (EQ MAKEA? T)
            (SETQ MAKEA? (AND (NULL MAKEA?)
                              (NOT (fetch ACKREQUESTED of SOCKET))
                              (PROG [(PUPALLOC (IMIN (fetch PUPALLOC of SOCKET)
                                                     (IQUOTIENT (fetch BYTEALLOC of SOCKET)
                                                                (fetch BYTESPERPUP of SOCKET]
                                                                    ; BCPL version also mins with socket allocations
                                    (RETURN (COND
                                              (\BSP.OVERLAP.DATA.WITH.ACK (ILEQ (LSH PUPALLOC 1)
                                                                                (fetch %#UNACKEDPUPS
                                                                                   of SOCKET)))
                                              (T (ILEQ PUPALLOC 0]
        (COND
           ((NOT (fetch ACKREQUESTED of SOCKET))                    ; unless ADATA is already outstanding, note the time so we can
                                                                    ; see how long partner takes to respond
            (SETUPTIMER 0 (fetch LASTADATATIME of SOCKET))
            (replace ACKREQUESTED of SOCKET with T)))
        (add (fetch ADATACOUNT of SOCKET)
             1)                                                     ; This is used to distinguish pups originally sent before this
                                                                    ; ADATA vs after
```

```
      ))
    (replace PUPTYPE of PUP with (SELECTC (fetch PUPTYPE of PUP)
                                      ((LIST \PT.DATA \PT.ADATA)
                                           (COND
                                              (MAKEA? \PT.ADATA)
                                              (T \PT.DATA)))
                                      ((LIST \PT.MARK \PT.AMARK)
                                           (COND
                                              (MAKEA? \PT.AMARK)
                                              (T \PT.MARK)))
                                      (BSPHELP "\TRANSMIT.STRATEGY called on non-data pup"])

)
```

;; BSP utilities

```
(DEFINEQ
```

(\BSP.DEFAULT.ERROR.HANDLER

```
  [LAMBDA (SOCKET ERRCODE)                                    (* bvm%: "11-AUG-81 12:30")
    (DECLARE (SPECVARS %#MYHANDLE#))                          ; Bind this to NIL to inhibit my toy scheduler
    (PROG (%#MYHANDLE#)
          (RETURN (ERROR (CONCAT "BSP error: " ERRCODE)
                         SOCKET])
```

(\BSP.TIMERFN

```
  [LAMBDA (SOCKET)                                            (* bvm%: " 8-Mar-84 17:53")
```

;;; Called when BSPTIMER expires. The timer gets reset every time we send something, so this means we haven't sent anything in a while

```
    (COND
       ((SELECTC (fetch RTPSTATE of SOCKET)
             ((LIST \STATE.OPEN \STATE.ENDSENT \STATE.ENDRECEIVED)
                 NIL)
             T)
        ;; Socket not alive, so kill it. CLOSERTPSOCKET will free up all resources except any waiting input, which will be held, I hope
        (CLOSERTPSOCKET SOCKET 0))
       ((AND (NOT \BSP.NO.INACTIVITY.TIMEOUT)
             (TIMEREXPIRED? (fetch INACTIVITYTIMER of SOCKET)))   ; Connection has fallen asleep, abort it
        (\RTP.ACTION SOCKET \EVENT.FORCECLOSE))
       (T (COND
             ((fetch ACKPENDING of SOCKET)                       ; I don't think this ever happens, because we can always get
                                                                 ; pups to do an ack with
              (\SEND.ACK SOCKET)))
          [PROG (PUP)
                (COND
                   ((AND (fetch INTERRUPTOUT of SOCKET)
                         (SETQ PUP (\SEARCH.OUTPUTQ SOCKET T))) ; Retransmit unacked interrupt
                    (replace EPREQUEUE of PUP with (fetch BSPOUTPUTQ of SOCKET))
                    (SENDPUP (fetch PUPSOC of SOCKET)
                             PUP]
          ;; Generate an ADATA unconditionally every BSPTIMER cycle, both to see whether partner is alive and to demonstrate that we are
          (COND
             ((NOT (fetch ACKREQUESTED of SOCKET))               ; ADATA not outstanding, so start timing
              (SETUPTIMER 0 (fetch LASTADATATIME of SOCKET))
              (replace ACKREQUESTED of SOCKET with T)))
          (add (fetch ADATACOUNT of SOCKET)
               1)
          (SENDPUP (fetch PUPSOC of SOCKET)
                   (\FILLBSPPUP SOCKET NIL \PT.ADATA \PUPOVLEN (fetch XMITBYTEID of SOCKET)))
          (\SETBSPTIMEOUT SOCKET])
```

(\BSP.FLUSH.SOCKET.QUEUES

```
  [LAMBDA (SOCKET)                                            (* bvm%: "26-OCT-83 14:51")
    (\FLUSH.PACKET.QUEUE (fetch BSPOUTPUTQ of SOCKET))
```

;; Flush anything waiting for output/retransmission. Don't flush input side, because someone might have a stream to keep reading from

```
    (PROG ((STREAM (fetch BSPINPUTSTREAM of SOCKET)))
          (OR STREAM (RETURN))
          (COND
             ((fetch BSPCURRENTPUP of (SETQ STREAM (fetch BSPOUTPUTSTREAM of STREAM)))
              (replace CBUFSIZE of STREAM with 0)
              (RELEASE.PUP (fetch BSPCURRENTPUP of STREAM))
              (replace BSPCURRENTPUP of STREAM with (replace CPPTR of STREAM with NIL])
```

(\FILLBSPPUP

```
  [LAMBDA (SOCKET PUP TYPE LENGTH ID REQUEUE)                 (* bvm%: " 1-JUL-83 12:23")
```

;;; Fills in the indicated fields of PUP, plus source and dest ports from SOCKET

```
    (OR PUP (SETQ PUP (ALLOCATE.PUP)))
```

```
    (replace PUPLENGTH of PUP with (OR LENGTH \PUPOVLEN))
    (replace TYPEWORD of PUP with TYPE)
    (replace PUPID of PUP with ID)
    (replace EPREQUEUE of PUP with (OR REQUEUE 'FREE))
    (\SETRTPPORTS SOCKET PUP)
    PUP])

(BSPHELP
  [LAMBDA (MSG)                                                    (* bvm%: " 5-JUN-83 15:51")
    (HELP "BSP error." MSG])

)

;; debugging

(DEFINEQ

(PPSOC
  [LAMBDA (SOC FILE VERBOSE)                                       ; Edited 6-Apr-87 20:03 by bvm:
    (PROG ((*STANDARD-OUTPUT* (\GETSTREAM FILE 'OUTPUT))
           (*PRINT-BASE* 10)
           [LOCALPORT (PORTSTRING (fetch LCLPORT of SOC)
                                  (\MAKENUMBER (fetch LCLSOCKETHI of SOC)
                                               (fetch LCLSOCKETLO of SOC]
           (STATE (fetch RTPSTATE of SOC))
           STREAM PUP)
          (if (EQ STATE \STATE.LISTENING)
              then (printout NIL "Listening on " LOCALPORT)
                   (RETURN))
          (printout NIL "From " LOCALPORT " to " (PORTSTRING (fetch FRNPORT of SOC)
                                                             (\MAKENUMBER (fetch FRNSOCKETHI of SOC)
                                                                          (fetch FRNSOCKETLO of SOC)))
                T "State: ")
          (PRINTCONSTANT STATE RTPSTATES)
          (printout NIL T "Last input activity: " .F1.1 (FQUOTIENT (+ (CLOCKDIFFERENCE (fetch INACTIVITYTIMER
                                                                                            of SOC))
                                                                      (fetch BSPINACTIVITYTIMEOUT of SOC))
                                                                   1000)
                " secs ago" T)
          (printout NIL "Connection id: " (fetch CONNID of SOC)
                T)
          (PRINTTIMER (fetch RTPTIMER of SOC)
                 (fetch RTPTIMEOUT of SOC)
                 "RTP timer: ")
          (PRINTTIMER (fetch BSPTIMER of SOC)
                 (fetch BSPTIMEOUT of SOC)
                 "BSP timer: ")
          (printout NIL T "Input:" T "  ID: " (fetch RCVBYTEID of SOC)
                 T "  InterruptID: " (fetch RCVINTERRUPTID of SOC)
                 T)
          [if (SETQ STREAM (fetch BSPINPUTSTREAM of SOC))
              then (printout NIL "  FilePtr: " (fetch BSPFILEPTR of STREAM)
                         T)
                   (COND
                      ((SETQ PUP (fetch BSPCURRENTPUP of STREAM))
                       (PPSOC.CURRENT STREAM PUP)
                       (COND
                          ((fetch MARKPENDING of STREAM)
                           (PRIN1 "{Mark pending}")))
                       (COND
                          (VERBOSE (TAB 4)
                                 (BSPPRINTPUP PUP T)
                                 (TERPRI]
          (PRINTPUPQUEUE (fetch BSPINPUTQ of SOC)
                 "  Input queue: " VERBOSE)
          (printout NIL "  #unread: " (fetch %#UNREADPUPS of SOC)
                 T)
          (printout NIL T "Output:" T "  ID: " (fetch XMITBYTEID of SOC)
                 T "  AckID: " (fetch LASTACKID of SOC)
                 T "  InterruptID: " (fetch XMITINTERRUPTID of SOC)
                 T "  Unacked pups: " (fetch %#UNACKEDPUPS of SOC)
                 ", bytes: "
                 (fetch %#UNACKEDBYTES of SOC)
                 T)
          [if (SETQ STREAM (fetch BSPOUTPUTSTREAM of STREAM))
              then (printout NIL "  FilePtr: " (fetch BSPFILEPTR of STREAM)
                         T)
                   (COND
                      ((SETQ PUP (fetch BSPCURRENTPUP of STREAM))
                       (PPSOC.CURRENT STREAM PUP]
          (PRINTPUPQUEUE (fetch BSPOUTPUTQ of SOC)
                 "  Retransmit queue: " VERBOSE)
          (printout NIL "  Alloc: " (fetch PUPALLOC of SOC)
                 " pups, "
                 (fetch BYTEALLOC of SOC)
                 " bytes, "
```

```
                (fetch BYTESPERPUP of SOC)
                "/pup" T "    Max " (fetch MAXPUPALLOC of SOC)
                ", cntr "
                (fetch PUPALLOCCOUNT of SOC)
                T)
          (printout NIL T "Flags: ")
          (COND
             ((fetch LISTENING of SOC)
              (PRIN1 "Listener, ")))
          (COND
             ((fetch INTERRUPTOUT of SOC)
              (PRIN1 "Interrupt out, ")))
          (COND
             ((fetch INTERRUPTIN of SOC)
              (PRIN1 "Interrupt in, ")))
          (COND
             ((fetch ACKPENDING of SOC)
              (PRIN1 "Ack pending, ")))
          (COND
             ((fetch ACKREQUESTED of SOC)
              (PRIN1 "Ack requested, ")))
          (COND
             ((fetch SENTZEROALLOC of SOC)
              (PRIN1 "Sent zero allocation.")))
          (TERPRI)
          (printout NIL "AData timeout: " (fetch ADATATIMEOUT of SOC])
```

(PPSOC.CURRENT

```
[LAMBDA (STREAM PUP)                                    (* bvm%: " 9-MAY-83 15:11")
  (printout NIL "  Current: " PUP " at " (fetch COFFSET of STREAM)
         ", "
         (IDIFFERENCE (fetch CBUFSIZE of STREAM)
                (fetch COFFSET of STREAM))
         " left" T])
```

(PRINTTIMER

```
[LAMBDA (TIMER TIMEOUT LABEL)                           (* bvm%: " 5-AUG-81 12:21")
  (COND
     ((AND TIMEOUT (NEQ TIMEOUT 0))
      (PRIN1 LABEL)
      (PROG ((DIF (IDIFFERENCE (CLOCKDIFFERENCE TIMER)
                        TIMEOUT)))
            (COND
               ((ILESSP DIF 0)
                (printout NIL (IMINUS DIF)
                       " msecs left" T))
               (T (printout NIL " expired " DIF " msecs ago." T])
```

(PRINTPUPQUEUE

```
[LAMBDA (QUEUE HEADER VERBOSE)                          (* bvm%: " 7-MAR-83 13:52")
  (PROG ((PUP (fetch SYSQUEUEHEAD of QUEUE))
         LASTPUP GAP)
        (PRIN1 HEADER)
        [COND
           (PUP (AND VERBOSE (TAB 4))
                (do (BSPPRINTPUP (SETQ LASTPUP PUP)
                           VERBOSE)
                   repeatwhile (AND (SETQ PUP (fetch QLINK of PUP))
                                    (PROGN (COND
                                              [VERBOSE (TAB 4)
                                                       (COND
                                                          ((NEQ (SETQ GAP (IDIFFERENCE (IDIFFERENCE (fetch PUPID
                                                                                                       of PUP)
                                                                                                    (fetch PUPID
                                                                                                       of LASTPUP))
                                                                                             (IDIFFERENCE (fetch PUPLENGTH
                                                                                                             of LASTPUP)
                                                                                                          \PUPOVLEN)))
                                                                0)
                                                           (printout NIL "<gap " GAP '> 4]
                                              (T (PRIN1 ", ")))
                                           T]
        (COND
           ((NEQ (fetch SYSQUEUETAIL of QUEUE)
                 LASTPUP)
            (printout NIL "  Oops! Tail of queue = " LASTPUP)))
        (TERPRI])
```

(BSPPRINTPUP

```
[LAMBDA (PUP VERBOSE)                                   (* bvm%: " 6-AUG-81 19:07")
  (COND
     ((NOT VERBOSE)
      (PRIN2 PUP))
```

```
       (T (printout NIL '{ (fetch PUPID of PUP)
                 " for "
                 (IDIFFERENCE (fetch PUPLENGTH of PUP)
                       \PUPOVLEN)
                 '})
          (SELECTC (fetch PUPTYPE of PUP)
              ((LIST \PT.AMARK \PT.MARK)
                  (printout NIL "[Mark " (GETBASEBYTE (fetch PUPCONTENTS of PUP)
                                                      0)
                         "]" T))
              (PRINTPUPDATA PUP '(CHARS])
```

(\RTP.INFO.HOOK

```
  [LAMBDA (PROC BUTTON)                                   (* bvm%: "10-JUL-83 22:25")
    (DECLARE (USEDFREE BSPSOCKET))                        ; This is evaluated underneath \RTP.SOCKET.PROCESS
    (PROG ((WINDOW (PROCESS.WINDOW PROC)))
          (COND
             ((NULL WINDOW)
              (SETQ WINDOW (CREATEW (GETBOXREGION 240 280)
                                    "BSP status"))
              (DSPFONT (FONTCREATE 'GACHA 8)
                       WINDOW)
              (PROCESS.WINDOW PROC WINDOW))
             (T (CLEARW WINDOW)))
          (PPSOC BSPSOCKET (WINDOWPROP WINDOW 'DSP)
                 (EQ BUTTON 'MIDDLE])

)

(DECLARE%: DONTCOPY

(ADDTOVAR PUPPRINTMACROS (8 BYTES 2 INTEGER)
                         (9 WORD 2 CHARS)
                         (16 CHARS)
                         (17 CHARS)
                         (18 WORDS)
                         (20 WORD 2 CHARS))
)

(/DECLAREDATATYPE 'BSPSOC
       '(WORD FIXP WORD FIXP BYTE POINTER POINTER POINTER POINTER POINTER WORD POINTER POINTER POINTER WORD
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER WORD POINTER POINTER POINTER WORD WORD
              POINTER WORD WORD WORD WORD WORD WORD POINTER WORD POINTER FLAG FLAG FLAG FLAG FLAG FLAG FLAG
              POINTER WORD POINTER POINTER POINTER POINTER POINTER POINTER POINTER)

       ;; ---field descriptor list elided by lister---

       '80)

(ADDTOVAR SYSTEMRECLST
          (DATATYPE BSPSOC ((FRNPORT WORD)
                            (FRNSOCKET FIXP)
                            (LCLPORT WORD)
                            (LCLSOCKET FIXP)
                            (RTPSTATE BYTE)
                            (RTPPROCESS POINTER)
                            (RTPEVENT POINTER)
                            (PUPSOC POINTER)
                            (CONNID POINTER)
                            (RTPTIMER POINTER)
                            (RTPTIMEOUT WORD)
                            (BSPINPUTHANDLER POINTER)
                            (BSPINPUTSTREAM POINTER)
                            (BSPTIMER POINTER)
                            (BSPTIMEOUT WORD)
                            (BSPFAILUREREASON POINTER)
                            (BSPOTHERPUPFN POINTER)
                            (BSPERRORHANDLER POINTER)
                            (BSPIOTIMEOUT POINTER)
                            (RCVBYTEID POINTER)
                            (RCVINTERRUPTID POINTER)
                            (BSPINPUTQ POINTER)
                            (%#UNREADPUPS WORD)
                            (XMITBYTEID POINTER)
                            (XMITINTERRUPTID POINTER)
                            (LASTACKID POINTER)
                            (%#UNACKEDPUPS WORD)
                            (%#UNACKEDBYTES WORD)
                            (BSPOUTPUTQ POINTER)
                            (BYTESPERPUP WORD)
                            (PUPALLOC WORD)
                            (BYTEALLOC WORD)
                            (MAXPUPALLOC WORD)
                            (PUPALLOCCOUNT WORD)
                            (ADATACOUNT WORD)
                            (LASTADATATIME POINTER)
                            (ADATATIMEOUT WORD)
```

```
                              (INACTIVITYTIMER POINTER)
                              (LISTENING FLAG)
                              (INTERRUPTOUT FLAG)
                              (INTERRUPTIN FLAG)
                              (ACKPENDING FLAG)
                              (ACKREQUESTED FLAG)
                              (SENTZEROALLOC FLAG)
                              (BSPNOACTIVITY FLAG)
                              (BSPUSERSTATE POINTER)
                              (NIL WORD)
                              (IOTIMEOUTFN POINTER)
                              (BSPWHENCLOSEDFN POINTER)
                              (BSPINPUTEVENT POINTER)
                              (BSPLOCK POINTER)
                              (BSPINITTIMER POINTER)
                              (BSPFAILURESTRING POINTER)
                              (BSPINACTIVITYTIMEOUT POINTER))))

(DECLARE%: DONTEVAL@LOAD DOCOPY

(\BSPINIT)
)

;; Some of these may want to be constants

(RPAQ? \BSPSOCKETS )

(RPAQ? \RFC.TIMEOUT 2000)

(RPAQ? \RTP.DALLY.TIMEOUT 5000)

(RPAQ? \RTP.DEFAULTTIMEOUT 30000)

(RPAQ? \BSP.MAXPUPS 12)

(RPAQ? \BSP.IDLETIMEOUT 15000)

(RPAQ? \BSP.OUTSTANDINGDATATIMEOUT 250)

(RPAQ? \BSP.MAXPUPALLOC 200)

(RPAQ? \BSP.ALLOCHYSTERESIS 50)

(RPAQ? \BSP.OVERLAP.DATA.WITH.ACK )

(RPAQ? \BSP.INITIAL.MAXPUPALLOC 5)

(RPAQ? \BSP.INITIAL.ADATATIMEOUT 1000)

(RPAQ? \BSP.MIN.ADATA.TIMEOUT 500)

(RPAQ? \BSP.MAX.ADATA.TIMEOUT 10000)

(RPAQ? \BSP.INACTIVITY.TIMEOUT 120000)

(RPAQ? \BSP.NO.INACTIVITY.TIMEOUT T)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \BSPSOCKETS \RFC.TIMEOUT \RTP.DALLY.TIMEOUT \RTP.DEFAULTTIMEOUT \BSP.MAXPUPS \BSP.IDLETIMEOUT
       \BSP.OUTSTANDINGDATATIMEOUT \BSP.MAXPUPALLOC \BSP.ALLOCHYSTERESIS \BSP.OVERLAP.DATA.WITH.ACK
       \BSP.INITIAL.MAXPUPALLOC \BSP.INITIAL.ADATATIMEOUT \BSP.MIN.ADATA.TIMEOUT \BSP.MAX.ADATA.TIMEOUT
       \BSP.INACTIVITY.TIMEOUT \BSP.NO.INACTIVITY.TIMEOUT)
)

(PUTPROPS BSP COPYRIGHT ("Venue & Xerox Corporation" 1982 1983 1900 1984 1985 1986 1987 1990 1993))
```

FUNCTION INDEX

VARIABLE INDEX

CONSTANT INDEX

MACRO INDEX

RECORD INDEX