Pilot

Volume 2 of 2

Date: October 1980
Version: 5.0

XEROX
Office Products Division
System Development Department
Palo Alto, California

```
-- PilotCounter.mesa (last edited by: Johnsson on: September 19, 1980 5:15 PM)

DIRECTORY
  CPSwapDefs USING [BBHandle, ExternalStateVector, UBBPointer],
  CountPrivate USING [ControlRecord, VersionID],
  Environment USING [wordsPerPage],
  Frame USING [GetReturnFrame, MyLocalFrame],
  Mopcodes USING [zKFCB, zPOP],
  PerformancePrograms USING [],
  PrincOps USING [FrameHandle, StateVector],
  ProcessInternal USING [DisableInterrupts, EnableInterrupts],
  ProcessOperations USING [ReadPSB],
  ProcessorFace USING [microsecondsPerHundredPulses],
  PSB USING [Handle, nullHandle, PDA],
  SDDefs USING [
    sBreakBlock, sCoreSwap, sGFTLength, SD, sXferTrap, sXferTrapMonitor],
  Space USING [Create, Handle, Map, mds, Pointer],
  SpecialSpace USING [MakeResident],
  System USING [GetClockPulses],
  XferTrap USING [WriteXTS];

PilotCounter: PROGRAM
  IMPORTS
    Frame, ProcessInternal, ProcessOperations, ProcessorFace, Space, SpecialSpace,
    System, XferTrap
  EXPORTS PerformancePrograms =

  BEGIN OPEN CountPrivate, Frame, PrincOps;

  SwapToDebugger: PROCEDURE [
    esv: LONG POINTER TO CPSwapDefs.ExternalStateVector] = MACHINE CODE
    BEGIN Mopcodes.zPOP; Mopcodes.zKFCB, SDDefs.sCoreSwap END;

  cr: ControlRecord;

  InitializePilotCounter: PUBLIC PROCEDURE =
    BEGIN
    sd: POINTER TO ARRAY [0..0) OF UNSPECIFIED ← SDDefs.SD;
    length: CARDINAL = MAX[sd[SDDefs.sGFTLength], 256];
    words: CARDINAL = length*SIZE[LONG CARDINAL]*2 + length;
    pages: CARDINAL =
      (words + Environment.wordsPerPage - 1)/Environment.wordsPerPage;
    data: POINTER;
    space: Space.Handle = Space.Create[parent: Space.mds, size: pages];
    Space.Map[space];
    SpecialSpace.MakeResident[space];
    data ← Space.Pointer[space];
    cr ←
      [gfi: 0, prevGfi: 0, version: VersionID, saveBreakHandler: NIL,
        length: length, newSession: TRUE, trace: FALSE, counts: data,
        times: data + length*SIZE[LONG CARDINAL], process: PSB.nullHandle,
        self: NIL, mode: plain, groups: data + 2*length*SIZE[LONG CARDINAL],
        newMeasurement: TRUE,
        pulseConversion: ProcessorFace.microsecondsPerHundredPulses];
    sd[SDDefs.sXferTrapMonitor] ← @cr;
    HandleTraps[];
    MonitorBreaks[];
    END;
```



```
  HandleTraps: PROCEDURE =
    BEGIN
    state: StateVector;
    frame: FrameHandle;
    finish, time, start: LONG CARDINAL;
    previousProcess: PSB.Handle;
    state ← STATE;
    state.dest ← GetReturnFrame[];
    SDDefs.SD[SDDefs.sXferTrap] ← state.source ← Frame.MyLocalFrame[];
    ProcessInternal.DisableInterrupts[];
    DO
      IF cr.trace THEN XferTrap.WriteXTS[skip1];
      previousProcess ← ProcessOperations.ReadPSB[];
      start ← System.GetClockPulses[];
      ProcessInternal.EnableInterrupts[];
      TRANSFER WITH state;
      ProcessInternal.DisableInterrupts[];
      state ← STATE;
      finish ← System.GetClockPulses[];
      XferTrap.WriteXTS[off];
      time ← finish - start;
      IF cr.newMeasurement THEN
        BEGIN cr.newMeasurement ← FALSE; cr.prevGfi ← 0; END
      ELSE
        IF cr.mode = plain THEN
          BEGIN
          cr.times.plain[cr.gfi] ← cr.times.plain[cr.gfi] + time;
          IF ProcessOperations.ReadPSB[] = previousProcess THEN
            cr.counts.plain[cr.gfi] ← cr.counts.plain[cr.gfi] + 1
          ELSE time ← time;
          END
        ELSE
          BEGIN
          to, from: [0..16);
          to ← cr.groups[cr.gfi];
          from ← cr.groups[cr.prevGfi];
          cr.prevGfi ← cr.gfi;
          cr.times.matrix[to][from] ← cr.times.matrix[to][from] + time;
          IF ProcessOperations.ReadPSB[] = previousProcess THEN
            cr.counts.matrix[to][from] ← cr.counts.matrix[to][from] + 1
          ELSE time ← time;
          END;
      state.dest ← frame ← GetReturnFrame[];
      cr.gfi ← frame.accesslink.gfi;
      IF
        (cr.process # PSB.nullHandle AND cr.process # ProcessOperations.ReadPSB[])
        THEN cr.gfi ← 0;
      ENDLOOP;
    END;

  MonitorBreaks: PROCEDURE =
    BEGIN
    state: StateVector;
    frame: FrameHandle;
    esv: LONG POINTER TO CPSwapDefs.ExternalStateVector ←
      PSB.PDA.externalStateVector;
    bbHandle: CPSwapDefs.BBHandle;
    ubb: CPSwapDefs.UBBPointer;
    i: CARDINAL;
    state ← STATE;
```

```
    cr.self ← MyLocalFrame[];
    state.dest ← GetReturnFrame[];
    ProcessInternal.DisableInterrupts[];
    DO
      ProcessInternal.EnableInterrupts[];
      TRANSFER WITH state;
      ProcessInternal.DisableInterrupts[];
      state ← STATE;
      state.dest ← frame ← GetReturnFrame[];
      bbHandle ← SDDefs.SD[SDDefs.sBreakBlock];
      FOR i IN [0..bbHandle.length) DO
        ubb ← @bbHandle.blocks[i];
        IF frame.accesslink = ubb.frame AND CARDINAL[frame.pc] = ubb.pc THEN
          BEGIN
          IF ubb.counterL THEN
            SELECT LOOPHOLE[ubb.ptrR, CARDINAL] FROM
              0 => cr.trace ← TRUE;
              1 => cr.trace ← ~cr.trace;
              2 => cr.trace ← FALSE;
              ENDCASE;
          state.instbyte ← ubb.inst;
          EXIT;
          END;
        REPEAT
          FINISHED =>
            BEGIN
            esv.state ← @state;
            esv.reason ← worrybreak;
            DO
              SwapToDebugger[esv];
              SELECT esv.reason FROM proceed => EXIT; ENDCASE;
              esv.reason ← return;
              ENDLOOP;
            END;
        ENDLOOP;
      IF cr.trace THEN XferTrap.WriteXTS[skip1]
      ELSE BEGIN cr.gfi ← cr.prevGfi ← 0; XferTrap.WriteXTS[off] END;
      ENDLOOP;
    END;

  StartCounting: PUBLIC PROCEDURE =
    BEGIN XferTrap.WriteXTS[skip1]; cr.trace ← TRUE; RETURN END;

  StopCounting: PUBLIC PROCEDURE =
    BEGIN cr.trace ← FALSE; cr.gfi ← cr.prevGfi ← 0; RETURN END;

  END.
LOG
Time: February 25, 1980 11:28 AM By: McJones Action: Change Kill to boot instea
**d of power off
Time: April 14, 1980 11:47 AM By: Knutsen Action: Module now STARTed by InitializePil
**otCounter[]
Time: April 24, 1980 11:49 AM By: McJones Action: Mesa 6
Time: July 29, 1980 9:04 AM By: McJones Action: Remove superseded kill, showscree
**n handlers from MonitorBreaks
Time: August 26, 1980 1:13 PM By: McJones Action: ProcessHandle = >Handle, CurrentP
**SB↑ = >ReadPSB[]; XferTrap
Time: September 19, 1980 10:05 AM By: Johnsson Action: XferTrap semantics
```

```
-- PilotPerfMonitor.Mesa (last edited by: September 19, 1980 11:10 AM)

DIRECTORY
  CPSwapDefs USING [ExternalStateVector, UBBPointer, UserBreakBlock],
  Frame USING [GetReturnFrame, MyLocalFrame],
  Inline USING [BITSHIFT],
  Mopcodes USING [zKFCB, zPOP, zRFS],
  PerfPrivate,
  PerformancePrograms USING [],
  PrincOps USING [BytePC, FieldDescriptor, FrameHandle, NullFrame, StateVector],
  ProcessInternal USING [DisableInterrupts, EnableInterrupts],
  ProcessOperations USING [ReadPSB],
  ProcessorFace USING [microsecondsPerHundredPulses],
  PSB USING [nullHandle, PDA],
  SDDefs USING [sBreakBlock, sBreakBlockSize, sCoreSwap, SD, sPerfMonitor],
  System USING [GetClockPulses],
  XferTrap USING [ReadXTS, Status, WriteXTS];

PilotPerfMonitor: PROGRAM
  IMPORTS
    Frame, Inline, ProcessInternal, ProcessOperations, ProcessorFace, System,
    XferTrap
  EXPORTS PerformancePrograms
  SHARES ProcessInternal =

  BEGIN OPEN PrincOps, PerfPrivate;

  SwapToDebugger: PROCEDURE [
    esv: LONG POINTER TO CPSwapDefs.ExternalStateVector] = MACHINE CODE
    BEGIN Mopcodes.zPOP; Mopcodes.zKFCB, SDDefs.sCoreSwap END;

  ReadField: PROCEDURE [POINTER, FieldDescriptor] RETURNS [UNSPECIFIED] = MACHINE
    CODE BEGIN Mopcodes.zRFS END;

  BreakBlock: TYPE = RECORD [
    count: CARDINAL, blocks: ARRAY [0..MaxNodes) OF CPSwapDefs.UserBreakBlock];

  perfRecord: PerfControlRecord;
  breakBlock: BreakBlock;
  nodeTab: POINTER TO NodeTab;
  legTab: POINTER TO LegTab;

  nodes: NodeTab;
  legs: LegTab;
  hists: ARRAY [0..HistSpaceSize) OF UNSPECIFIED ← ALL[0];

  InitializePilotPerfMonitor: PUBLIC PROCEDURE =
    BEGIN
    i: CARDINAL;
    perfRecord ← PerfControlRecord[
      version: VersionID, measuringNow: FALSE, newSession: TRUE, addLeg: none,
      trackLeg: all, totalBreaks: 0, perfTime: 0, totalTime: 0,
      self: PrincOps.NullFrame, saveBreakHandler: PrincOps.NullFrame, nextNode: 0,
      process: PSB.nullHandle, nodeTable: @nodes, nextLeg: 0, legTable: @legs,
      lastID: NullNode, histBase: @hists, lastCall: normal,
      histFree: FIRST[HistIndex],
      pulseConversion: ProcessorFace.microsecondsPerHundredPulses];
    SDDefs.SD[SDDefs.sPerfMonitor] ← @perfRecord;
```



```
    SDDefs.SD[SDDefs.sBreakBlock] ← @breakBlock;
    SDDefs.SD[SDDefs.sBreakBlockSize] ← SIZE[BreakBlock];
    breakBlock.count ← 0;
    FOR i IN [0..MaxNodes) DO
      nodes[i] ←
        [id: NullID, hitsLow: 0, hitsHigh: 0, overflowed: FALSE, hist: NullHist];
      ENDLOOP;
    FOR i IN [0..MaxLegs) DO
      legs[i] ←
        [start: 0, from: NullNode, to: NullNode, owner: PSB.nullHandle,
          hitsLow: 0, hitsHigh: 0, sum: 0, lock: FALSE, overflowed: FALSE,
          someIgnored: FALSE, hist: NullHist];
      ENDLOOP;
    nodeTab ← @nodes;
    legTab ← @legs;

    MonitorBreaks[];
    END;

  MonitorBreaks: PROCEDURE =
    BEGIN OPEN Inline;
    state: StateVector;
    frame: FrameHandle;
    esv: LONG POINTER TO CPSwapDefs.ExternalStateVector ←
      PSB.PDA.externalStateVector;
    xferTrapStatus: XferTrap.Status;
    ubb: CPSwapDefs.UBBPointer;
    pCR: POINTER TO PerfControlRecord;
    locL: POINTER;
    fd: FieldDescriptor;
    id: CARDINAL;
    value: CARDINAL;
    word: INTEGER;
    longval: LongNumber;
    hist: POINTER TO Histogram;
    timeSpent: LongNumber;
    timeOnEntry: Number;
    lastExit: Number;
    lastEntry: Number;
    lastPerfEntry: Number;
    lastOverhead: Number;
    lastLegTime: Number;
    breakType: {perf, normal, other};
    stillMustDoLastLeg: BOOLEAN;
    i: CARDINAL;
    free: BOOLEAN;
    node: POINTER TO Node;
    leg: POINTER TO Leg;
    nodeID: NodeID;
    state ← STATE;
    pCR ← @perfRecord;
    pCR.self ← Frame.MyLocalFrame[];
    state.dest ← Frame.GetReturnFrame[];
    ProcessInternal.DisableInterrupts[];
    DO
      OPEN pCR;
      xferTrapStatus ← XferTrap.ReadXTS[];
      IF xferTrapStatus = on THEN XferTrap.WriteXTS[skip1];
```

```
    lastEntry ← timeOnEntry;
    lastExit ← System.GetClockPulses[];
    ProcessInternal.EnableInterrupts[];
    TRANSFER WITH state;
    ProcessInternal.DisableInterrupts[];
    state ← STATE;
    timeOnEntry ← System.GetClockPulses[];
    state.dest ← frame ← Frame.GetReturnFrame[];


    IF ~measuringNow THEN
      BEGIN lastEntry ← lastExit; lastPerfEntry ← timeOnEntry; END;
    lastOverhead ← lastExit - lastEntry;
    lastPerfEntry ← lastPerfEntry + lastOverhead;
    IF lastCall = perf THEN
      BEGIN
      perfTime ← perfTime + lastOverhead;
      totalTime ← totalTime + lastOverhead;
      END
    ELSE
      FOR leg ← @legTab[0], leg + SIZE[Leg] UNTIL leg = @legTab[nextLeg] DO
        IF leg.owner # PSB.nullHandle THEN leg.start ← leg.start + lastOverhead;
        ENDLOOP;
    nodeID ← [pc: PrincOps.BytePC[frame.pc], frame: frame.accesslink];
    breakType ← normal;
    FOR i IN [0..breakBlock.count) DO
      ubb ← @breakBlock.blocks[i];
      IF nodeID = [frame: ubb.frame, pc: ubb.pc] THEN
        BEGIN
        state.instbyte ← ubb.inst;
        node ← @nodeTab[0];
        IF process # PSB.nullHandle AND process # ProcessOperations.ReadPSB[]
          THEN BEGIN breakType ← other; EXIT END;
        FOR id IN [0..nextNode) DO
          IF nodeID = node.id THEN GOTO foundEntry;
          node ← node + SIZE[Node];
          ENDLOOP;
        IF nextNode < MaxNodes THEN
          BEGIN
          id ← nextNode;
          nextNode ← nextNode + 1;
          node↑ ←
            [id: nodeID, hitsLow: 0, hitsHigh: 0, overflowed: FALSE,
              hist: NullHist];
          GO TO foundEntry;
          END
        ELSE BEGIN breakType ← other; EXIT END;
        END;
      REPEAT
        foundEntry =>
          BEGIN
          breakType ← perf;
          IF (node.hitsLow ← node.hitsLow + 1) = 0 THEN
            IF (node.hitsHigh ← node.hitsHigh + 1) = 0 THEN
              node.overflowed ← TRUE;
          IF node.hist # NullHist AND ~ubb.counterL THEN
            BEGIN
            locL ←
              IF ubb.localL THEN frame + LOOPHOLE[ubb.ptrL, CARDINAL]
              ELSE ubb.ptrL;
```



```
            fd ← [offset: 0, posn: ubb.posnL, size: ubb.sizeL];
            value ← ReadField[locL, fd];
            hist ← @histBase[node.hist];
            hist.count ← hist.count + 1;
            hist.sum ← hist.sum + value;
            IF LOOPHOLE[hist.base, LongNumber].lowbits > value THEN
              hist.underflow ← hist.underflow + 1
            ELSE
              BEGIN
              value ←
                (value - LOOPHOLE[hist.base, LongNumber].lowbits)/hist.scale;
              IF hist.class = log THEN
                IF (word ← value) # 0 THEN
                  FOR value DECREASING IN [0..15] DO
                    IF word < 0 THEN EXIT; word ← word*2; ENDLOOP;
              IF value < hist.nBuckets THEN
                hist.buckets[value] ← hist.buckets[value] + 1
              ELSE hist.overflow ← hist.overflow + 1;
              END;
            END;
          END;
      ENDLOOP;


    IF breakType = perf THEN
      BEGIN
      lastLegTime ← timeOnEntry - lastPerfEntry;
      totalTime ← totalTime + lastLegTime;
      IF trackLeg # none THEN
        BEGIN
        stillMustDoLastLeg ← TRUE;
        FOR leg ← @legTab[0], leg + SIZE[Leg] UNTIL leg = @legTab[nextLeg] DO
          IF leg.owner # PSB.nullHandle THEN
            BEGIN
            leg.start ← leg.start + lastOverhead;
            IF id = leg.to THEN
              BEGIN
              IF leg.owner # ProcessOperations.ReadPSB[] THEN
                leg.someIgnored ← TRUE
              ELSE
                BEGIN
                leg.owner ← PSB.nullHandle;
                IF lastID = leg.from THEN stillMustDoLastLeg ← FALSE;
                IF (leg.hitsLow ← leg.hitsLow + 1) = 0 THEN
                  IF (leg.hitsHigh ← leg.hitsHigh + 1) = 0 THEN
                    leg.overflowed ← TRUE;
                timeSpent.lc ← timeOnEntry - leg.start;
                leg.sum ← leg.sum + timeSpent.lc;
                IF leg.hist # NullHist THEN
                  BEGIN
                  hist ← @histBase[leg.hist];
                  hist.count ← hist.count + 1;
                  hist.sum ← hist.sum + timeSpent.lc;
                  IF hist.base > timeSpent.lc THEN
                    hist.underflow ← hist.underflow + 1
                  ELSE
                    BEGIN
                    timeSpent.lc ← timeSpent.lc - hist.base;
                    longval.highbits ← BITSHIFT[
```

```
                timeSpent.highbits, -hist.scale];
              longval.lowbits ←
                BITSHIFT[timeSpent.lowbits, -hist.scale] + BITSHIFT[
                  timeSpent.highbits, 16 - hist.scale];
              IF hist.class = log THEN
                IF (word ← longval.highbits) # 0 THEN
                  FOR value DECREASING IN [16..31] DO
                    IF word < 0 THEN EXIT; word ← word*2; ENDLOOP
                ELSE
                  IF (word ← longval.lowbits) # 0 THEN
                    FOR value DECREASING IN [0..15] DO
                      IF word < 0 THEN EXIT; word ← word*2; ENDLOOP
                  ELSE value ← 0
              ELSE
                value ←
                  IF longval.highbits # 0 THEN hist.nBuckets
                  ELSE longval.lowbits;
              IF value < hist.nBuckets THEN
                hist.buckets[value] ← hist.buckets[value] + 1
              ELSE hist.overflow ← hist.overflow + 1;
              END;
            END;
          END;
        END
      ELSE IF trackLeg = successor THEN leg.owner ← PSB.nullHandle;
      END;
    IF id = leg.from THEN
      BEGIN
      leg.start ← timeOnEntry;
      IF leg.owner # PSB.nullHandle THEN leg.someIgnored ← TRUE;
      leg.owner ← ProcessOperations.ReadPSB[];
      END;
    ENDLOOP;
  free ← FALSE;
  IF addLeg = successor AND stillMustDoLastLeg AND measuringNow THEN
    BEGIN
    FOR leg ← @legTab[0], leg + SIZE[Leg] UNTIL leg = @legTab[nextLeg] DO
      IF ~leg.lock AND leg.from = NullNode THEN
        BEGIN free ← TRUE; EXIT; END;
      ENDLOOP;
    IF ~free AND nextLeg < MaxLegs THEN
      BEGIN
      leg ← @legTab[nextLeg];
      nextLeg ← nextLeg + 1;
      free ← TRUE
      END;
    IF free THEN
      BEGIN
      leg↑ ←
        [start: timeOnEntry, from: lastID, to: id, lock: FALSE,
          someIgnored: FALSE, owner: PSB.nullHandle, hitsLow: 1,
          hitsHigh: 0, sum: lastLegTime, hist: NullHist,
          overflowed: FALSE];
      IF id = lastID THEN leg.owner ← ProcessOperations.ReadPSB[];
      stillMustDoLastLeg ← FALSE;
      END;
    END;
  END;
END;
```



```
    SELECT breakType FROM
      perf =>
        BEGIN
        lastID ← id;
        measuringNow ← TRUE;
        totalBreaks ← totalBreaks + 1;
        lastPerfEntry ← timeOnEntry;
        lastCall ← perf;
        END;
      normal =>
        BEGIN
        esv.state ← @state;
        esv.reason ← worrybreak;
        DO
          SwapToDebugger[esv];
          -- now subtract out time spent in the debugging world
          lastCall ← normal;
          SELECT esv.reason FROM proceed => EXIT; ENDCASE;
          esv.reason ← return;
          ENDLOOP;
        END;
      ENDCASE => lastCall ← normal;

    ENDLOOP;
  END;

END.
```

July 29, 1980 9:03 AM McJones Remove superseded kill, showscreen handlers of MonitorBreaks
September 15, 1980 3:51 PM McJones XferTrap
September 19, 1980 10:04 AM Johnsson PrincOps breaks; mergen monitor and break handler m
***odules*

```
-- HeapImpl.mesa (last edited by: McJones on: August 5, 1980  1:54 PM)

DIRECTORY
  Environment USING [maxPagesInMDS, PageCount, wordsPerPage],
  Heap USING [Create, CreateMDS, defaultSwapUnit, ErrorType, NWords],
  Inline USING [LowHalf],
  MiscPrograms USING [],
  PilotSwitches USING [switches --.u--],
  Runtime USING [GetCaller],
  Space USING [
    Create, CreateUniformSwapUnits, Delete, Error, GetAttributes, GetHandle,
    Handle, InsufficientSpace, LongPointer, Map, mds, nullHandle, PageCount,
    PageFromLongPointer, virtualMemory],
  SpecialHeap USING [],
  SpecialSpace USING [MakeResident, MakeSwappable],
  SystemInternal USING [Unimplemented],
  Zone USING [
    AddSegment, Base, Create, FreeNode, GetAttributes, GetSegmentAttributes,
    Handle, MakeNode, minimumNodeSize, nullSegment, RemoveSegment, SegmentHandle,
    SetChecking, Status],
  ZoneInternal USING [NodeHeader, ZoneHeader];

HeapImpl: MONITOR
  IMPORTS
    Heap, Inline, PilotSwitches, Runtime, Space, SpecialSpace, SystemInternal,
    Zone
  EXPORTS Heap, MiscPrograms, SpecialHeap =
  BEGIN

  --
  -- Types

  NWords: TYPE = Heap.NWords;

  UncountedZoneRep: TYPE = LONG POINTER TO UncountedZoneObject;
  UncountedZoneObject: TYPE = MACHINE DEPENDENT RECORD [
    procs(0:0..31): LONG POINTER TO Procs, data(2:0..31): Handle];
  Procs: TYPE = MACHINE DEPENDENT RECORD [
    Make(0): PROCEDURE [UNCOUNTED ZONE, NWords] RETURNS [LONG POINTER],
    Free(1): PROCEDURE [UNCOUNTED ZONE, LONG POINTER]
    -- could add other generic Heap operations here
    ];
  Handle: TYPE = LONG POINTER TO Data;

  MDSZoneRep: TYPE = POINTER TO MDSZoneObject;
  MDSZoneObject: TYPE = MACHINE DEPENDENT RECORD [
    procs(0:0..15): POINTER TO MDSProcs, data(1:0..15): MDSHandle];
  MDSProcs: TYPE = MACHINE DEPENDENT RECORD [
    Make(0): PROCEDURE [MDSZone, NWords] RETURNS [POINTER],
    Free(1): PROCEDURE [MDSZone, POINTER]
    -- could add other generic (MDS) Heap operations here
    ];
  MDSHandle: TYPE = POINTER TO Data;

  Data: TYPE = RECORD [
    seal: INTEGER ← currentSeal,
    parent: Space.Handle,
    base: Zone.Base, -- = Space.LongPointer[parent]
```



```
    zH: Zone.Handle,
    resident: BOOLEAN ← FALSE,
    ownerChecking, checking: BOOLEAN,
    increment, swapUnit: Environment.PageCount,
    threshold, largeNodeThreshold: NWords,
    largeNodes: LONG POINTER TO LargeNodeHeader ← NIL,
    vp:
      SELECT OVERLAID * FROM
        normal => [uzo: UncountedZoneObject],
        mds => [mzo: MDSZoneObject],
        ENDCASE];

  LargeNodeHeader: TYPE = MACHINE DEPENDENT RECORD [
    next(0): LONG POINTER TO LargeNodeHeader,
    nodeHeader(2): inuse ZoneInternal.NodeHeader,
    node(3): ARRAY [0..0) OF RECORD [UNSPECIFIED]];

  --
  -- Constants

  currentSeal: INTEGER = 12345;
  defaultSwapUnit: Space.PageCount = Heap.defaultSwapUnit;
  pMDS: LONG POINTER = Space.LongPointer[Space.mds];
  nodeOverhead: CARDINAL = SIZE[inuse ZoneInternal.NodeHeader];
  zoneOverhead: CARDINAL = SIZE[ZoneInternal.ZoneHeader] + nodeOverhead;
  minimumNodeSize: PUBLIC NWords = Zone.minimumNodeSize;
  initial: Environment.PageCount ← 2;
  initialMDS: Environment.PageCount ← 2;
  standardProcs: Procs ← [Make: MakeNode, Free: FreeNode];
  standardMDSProcs: MDSProcs ← [Make: MakeMDSNode, Free: FreeMDSNode];
  hyperSpace: Space.Handle; -- initialized in InitializeHeap
  systemZone: PUBLIC UNCOUNTED ZONE; -- initialized in InitializeHeap
  systemMDSZone: PUBLIC MDSZone = Heap.CreateMDS[initial: initialMDS];

  --
  -- Initialization (MiscPrograms)

  InitializeHeap: PUBLIC PROCEDURE =
    BEGIN
    -- Because UtilityPilot currently only allows one level of spaces, with parent equal to
    -- virtualMemory or mds, we can't support general "hyper" heaps.  We make a special
    -- case of systemZone, by identifying it with systemMDSZone.
    IF PilotSwitches.switches.u = down THEN -- UtilityPilot
      BEGIN
      uz: UncountedZoneRep = systemMDSZone.NEW[
        UncountedZoneObject ←
          [procs: @standardProcs,
            data: LOOPHOLE[systemMDSZone, MDSZoneRep].data]];
      hyperSpace ← Space.nullHandle; -- Heap.Create unimplemented in UtilityPilot
      systemZone ← LOOPHOLE[uz];
      END
    ELSE
      BEGIN
      hyperSpace ← Space.Create[
        parent: Space.virtualMemory, size: Environment.maxPagesInMDS];
      systemZone ← Heap.Create[initial: initial]
      END;
    END;
```

```
--
-- Heap implementation

Create: PUBLIC PROCEDURE [
  initial: Environment.PageCount, parent: Space.Handle,
  increment, swapUnit: Environment.PageCount,
  threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN]
  RETURNS [UNCOUNTED ZONE] =
  BEGIN
  h: Handle;
  IF PilotSwitches.switches.u = down THEN ERROR SystemInternal.Unimplemented;
  IF parent = Space.nullHandle THEN parent ← hyperSpace;
  h ← CreateHeap[
    initial, parent, increment, swapUnit, threshold, largeNodeThreshold,
    ownerChecking, checking];
  h.uzo ← [procs: @standardProcs, data: h];
  RETURN[LOOPHOLE[@h.uzo]]
  END;

CreateMDS: PUBLIC PROCEDURE [
  initial: Environment.PageCount, increment, swapUnit: Environment.PageCount,
  threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN]
  RETURNS [MDSZone] =
  BEGIN
  h: MDSHandle = Narrow[
    CreateHeap[
    initial, Space.mds, increment, swapUnit, threshold, largeNodeThreshold,
    ownerChecking, checking]];
  h.mzo ← [procs: @standardMDSProcs, data: h];
  RETURN[LOOPHOLE[@h.uzo]]
  END;

CreateHeap: PROCEDURE [
  initial: Environment.PageCount, parent: Space.Handle,
  increment, swapUnit: Environment.PageCount,
  threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN]
  RETURNS [Handle] =
  BEGIN
  zH: Zone.Handle;
  status: Zone.Status;
  heap: Zone.Base RELATIVE POINTER TO Data;
  base: Zone.Base;
  seg: LONG POINTER;
  IF initial = 0 THEN initial ← 1; -- since not yet ready to extend
  base ← Space.LongPointer[parent];
  seg ← MakeSpace[
    parent: parent, pages: initial, swapUnit: swapUnit, resident: FALSE !
    Space.InsufficientSpace => GOTO InsufficientSpace];
  [zH: zH, s: status] ← Zone.Create[
    storage: seg, length: initial*Environment.wordsPerPage, zoneBase: base,
    threshold: threshold, checking: checking];
  IF status ~= okay THEN ERROR Error[otherError];
  [heap, status] ← Zone.MakeNode[zH: zH, n: SIZE[Data]];
  IF status ~= okay THEN ERROR Error[otherError];
  base[heap] ←
    [parent: parent, base: base, zH: zH, ownerChecking: ownerChecking,
      checking: checking, increment: increment, swapUnit: swapUnit,
      threshold: threshold, largeNodeThreshold: largeNodeThreshold, vp: NULL];
```



```
  RETURN[@base[heap]]
  EXITS InsufficientSpace => ERROR Error[insufficientSpace]
  END;

Delete: PUBLIC PROCEDURE [z: UNCOUNTED ZONE, checkEmpty: BOOLEAN] =
  BEGIN OPEN rep: LOOPHOLE[z, UncountedZoneRep];
  DeleteHeap[rep.data, checkEmpty];
  END;

DeleteMDS: PUBLIC PROCEDURE [z: MDSZone, checkEmpty: BOOLEAN] =
  BEGIN OPEN rep: LOOPHOLE[z, MDSZoneRep];
  DeleteHeap[rep.data, checkEmpty];
  END;

DeleteHeap: PROCEDURE [h: Handle, checkEmpty: BOOLEAN] =
  BEGIN
  DeleteSpace: PROCEDURE [
    sH: Space.Handle, isLargeNode: BOOLEAN, segH: Zone.SegmentHandle] =
    BEGIN
    IF checkEmpty AND
      (isLargeNode OR segH ~= Zone.nullSegment AND Zone.RemoveSegment[
        h.zH, segH].s ~= okay) THEN ERROR Error[invalidHeap];
    Space.Delete[sH];
    END;
  EnumerateSpaces[h, DeleteSpace];
  END;

MakeNode: PUBLIC PROCEDURE [z: UNCOUNTED ZONE, n: NWords]
  RETURNS [p: LONG POINTER] =
  BEGIN
  h: Handle = LOOPHOLE[z, UncountedZoneRep].data;
  r: Zone.Base RELATIVE POINTER;
  status: Zone.Status;
  IF h.ownerChecking THEN n ← n + 1;
  IF n >= h.largeNodeThreshold THEN p ← MakeLargeNode[h, n]
  ELSE
    DO
      [node: r, s: status] ← Zone.MakeNode[zH: h.zH, n: n];
      SELECT status FROM
        okay => BEGIN p ← @h.base[r]; EXIT END;
        noRoomInZone => ExpandHeap[h, PagesForNewSegment[n]];
        ENDCASE => ERROR Error[invalidHeap];
      ENDLOOP;
  IF h.ownerChecking THEN BEGIN p↑ ← Runtime.GetCaller[]; p ← p + 1 END;
  END;

FreeNode: PUBLIC PROCEDURE [z: UNCOUNTED ZONE, p: LONG POINTER] =
  BEGIN
  h: Handle = LOOPHOLE[z, UncountedZoneRep].data;
  IF h.ownerChecking THEN p ← p - 1;
  IF
    LOOPHOLE[p - SIZE[inuse ZoneInternal.NodeHeader], LONG POINTER TO
      ZoneInternal.NodeHeader].length - nodeOverhead < h.largeNodeThreshold THEN
    SELECT Zone.FreeNode[zH: h.zH, p: p] FROM -- surely regular node

      okay => NULL;
      invalidNode => ERROR Error[invalidNode];
      ENDCASE => ERROR Error[invalidHeap]
  ELSE FreeLargeOrRegularNode[h, p] -- possibly large node
```

```
  END;

MakeMDSNode: PUBLIC PROCEDURE [z: MDSZone, n: NWords] RETURNS [p: POINTER] =
  BEGIN
  h: Handle = LOOPHOLE[z, MDSZoneRep].data;
  status: Zone.Status;
  IF h.ownerChecking THEN n ← n + 1;
  -- since we can't delete space in UtilityPilot, treat large as small
  IF n >= h.largeNodeThreshold AND PilotSwitches.switches.u = up THEN
    p ← Narrow[MakeLargeNode[h, n]]
  ELSE
    DO
      [node: LOOPHOLE[p, Zone.Base RELATIVE POINTER], s: status] ←
        Zone.MakeNode[zH: h.zH, n: n];
      SELECT status FROM
        okay => EXIT;
        noRoomInZone => ExpandHeap[h, PagesForNewSegment[n]];
        ENDCASE => ERROR Error[invalidHeap]
      ENDLOOP;
  IF h.ownerChecking THEN BEGIN p↑ ← Runtime.GetCaller[]; p ← p + 1 END;
  END;

FreeMDSNode: PUBLIC PROCEDURE [z: MDSZone, p: POINTER] =
  BEGIN
  h: Handle = LOOPHOLE[z, MDSZoneRep].data;
  IF h.ownerChecking THEN p ← p - 1;
  IF
    LOOPHOLE[p - SIZE[inuse ZoneInternal.NodeHeader], POINTER TO
      ZoneInternal.NodeHeader].length - nodeOverhead < h.largeNodeThreshold THEN
    SELECT Zone.FreeNode[zH: h.zH, p: p] FROM -- surely regular node

      okay => NULL;
      invalidNode => ERROR Error[invalidNode];
      ENDCASE => ERROR Error[invalidHeap]
  ELSE FreeLargeOrRegularNode[h, p] -- possibly large node

  END;

GetAttributes: PUBLIC PROCEDURE [z: UNCOUNTED ZONE]
  RETURNS [
    parent: Space.Handle,
    heapPages, largeNodePages, increment, swapUnit: Environment.PageCount,
    threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN] =
  BEGIN OPEN data: LOOPHOLE[z, UncountedZoneRep].data;
  [parent: parent, heapPages: heapPages, largeNodePages: largeNodePages,
    increment: increment, swapUnit: swapUnit, threshold: threshold,
    largeNodeThreshold: largeNodeThreshold, ownerChecking: ownerChecking] ←
    GetAttributesHeap[@data];
  END;

GetAttributesMDS: PUBLIC PROCEDURE [z: MDSZone]
  RETURNS [
    heapPages, largeNodePages, increment, swapUnit: Environment.PageCount,
    threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN] =
  BEGIN OPEN data: LOOPHOLE[z, MDSZoneRep].data;
  [parent:, -- = Space.mds
    heapPages: heapPages, largeNodePages: largeNodePages, increment: increment,
    swapUnit: swapUnit, threshold: threshold,
    largeNodeThreshold: largeNodeThreshold, ownerChecking: ownerChecking] ←
```



```
    GetAttributesHeap[@data];
  END;

GetAttributesHeap: PROCEDURE [h: Handle]
  RETURNS [
    parent: Space.Handle,
    heapPages, largeNodePages, increment, swapUnit: Environment.PageCount,
    threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN] =
  BEGIN
  CountSpace: PROCEDURE [
    sH: Space.Handle, isLargeNode: BOOLEAN, segH: Zone.SegmentHandle] =
    BEGIN
    pages: Environment.PageCount = Space.GetAttributes[sH].size;
    IF isLargeNode THEN largeNodePages ← largeNodePages + pages
    ELSE -- main or extension segment --heapPages ← heapPages + pages;
    END;
  heapPages ← largeNodePages ← 0;
  EnumerateSpaces[h, CountSpace];
  RETURN[
    parent: IF h.parent = hyperSpace THEN Space.nullHandle ELSE h.parent,
      heapPages: heapPages, largeNodePages: largeNodePages,
      increment: h.increment, swapUnit: h.swapUnit, threshold: h.threshold,
      largeNodeThreshold: h.largeNodeThreshold, ownerChecking: h.ownerChecking,
      checking: h.checking]
  END;

Expand: PUBLIC PROCEDURE [z: UNCOUNTED ZONE, pages: Space.PageCount] =
  BEGIN ExpandHeap[LOOPHOLE[z, UncountedZoneRep].data, pages]; END;

ExpandMDS: PUBLIC PROCEDURE [z: MDSZone, pages: Space.PageCount] =
  BEGIN ExpandHeap[LOOPHOLE[z, MDSZoneRep].data, pages]; END;

ExpandHeap: PROCEDURE [h: Handle, pages: Space.PageCount] =
  -- Add a segment of the specified size (or h.increment if that is larger) to h.
  BEGIN
  status: Zone.Status;
  pages ← MAX[pages, h.increment];
  [sH:, s: status] ← Zone.AddSegment[
    zH: h.zH,
    storage: MakeSpace[
    h.parent, pages, h.swapUnit, h.resident !
    Space.InsufficientSpace => GOTO InsufficientSpace;
    Space.Error => GOTO UnsuitableParent],
    length: pages*Environment.wordsPerPage];
  IF status ~= okay THEN ERROR Error[otherError];
  EXITS
    InsufficientSpace => RETURN WITH ERROR Error[insufficientSpace];
    UnsuitableParent => RETURN WITH ERROR Error[unsuitableParent];
  END;

Prune: PUBLIC PROCEDURE [z: UNCOUNTED ZONE] =
  BEGIN PruneHeap[LOOPHOLE[z, UncountedZoneRep].data]; END;

PruneMDS: PUBLIC PROCEDURE [z: MDSZone] =
  BEGIN PruneHeap[LOOPHOLE[z, MDSZoneRep].data]; END;

PruneHeap: PROCEDURE [h: Handle] =
  BEGIN
  PruneSegment: PROCEDURE [
```

```
    sH: Space.Handle, isLargeNode: BOOLEAN, segH: Zone.SegmentHandle] =
    BEGIN
    IF ~isLargeNode AND segH ~= Zone.nullSegment THEN
      -- only interested in extension segments
      SELECT Zone.RemoveSegment[h.zH, segH].s FROM
        okay => Space.Delete[sH];
        nonEmptySegment => NULL;
        ENDCASE => ERROR Error[invalidHeap];
    END;
  EnumerateSpaces[h, PruneSegment];
  END;

CheckOwner: PUBLIC PROCEDURE [p: LONG POINTER] =
  BEGIN IF (p - 1)↑ ~= Runtime.GetCaller[] THEN ERROR Error[invalidOwner]; END;

SetChecking: PUBLIC PROCEDURE [z: UNCOUNTED ZONE, checking: BOOLEAN] =
  BEGIN SetCheckingHeap[LOOPHOLE[z, UncountedZoneRep].data, checking]; END;

SetCheckingMDS: PUBLIC PROCEDURE [z: MDSZone, checking: BOOLEAN] =
  BEGIN SetCheckingHeap[LOOPHOLE[z, MDSZoneRep].data, checking]; END;

SetCheckingHeap: PROCEDURE [h: Handle, checking: BOOLEAN] =
  BEGIN
  IF Zone.SetChecking[zH: h.zH, checking: h.checking ← checking].s ~= okay THEN
    ERROR Error[invalidHeap];
  END;

Error: PUBLIC ERROR [type: Heap.ErrorType] = CODE;

--
-- SpecialHeap procedures

MakeResident: PUBLIC PROCEDURE [z: UNCOUNTED ZONE] =
  BEGIN MakeResidentHeap[LOOPHOLE[z, UncountedZoneRep].data]; END;

MakeResidentMDS: PUBLIC PROCEDURE [z: MDSZone] =
  BEGIN MakeResidentHeap[LOOPHOLE[z, MDSZoneRep].data]; END;

MakeResidentHeap: PROCEDURE [h: Handle] =
  BEGIN
  MakeResident1: PROCEDURE [
    sH: Space.Handle, isLargeNode: BOOLEAN, segH: Zone.SegmentHandle] =
    BEGIN SpecialSpace.MakeResident[sH] END;
  EnumerateSpaces[h, MakeResident1];
  h.resident ← TRUE;
  END;

MakeSwappable: PUBLIC PROCEDURE [z: UNCOUNTED ZONE] =
  BEGIN MakeSwappableHeap[LOOPHOLE[z, UncountedZoneRep].data]; END;

MakeSwappableMDS: PUBLIC PROCEDURE [z: MDSZone] =
  BEGIN MakeSwappableHeap[LOOPHOLE[z, MDSZoneRep].data]; END;

MakeSwappableHeap: PROCEDURE [h: Handle] =
  BEGIN
  MakeSwappable1: PROCEDURE [
    sH: Space.Handle, isLargeNode: BOOLEAN, segH: Zone.SegmentHandle] =
    BEGIN SpecialSpace.MakeSwappable[sH] END;
```



```
  EnumerateSpaces[h, MakeSwappable1];
  h.resident ← FALSE;
  END;

--
-- Miscellaneous

EnumerateSpaces: PROCEDURE [
  h: Handle,
  P: PROCEDURE [
    sH: Space.Handle, isLargeNode: BOOLEAN, segH: Zone.SegmentHandle]] =
  -- Call.P once for each space of heap h. There is one space for:
  --   each large node (isLargeNode = TRUE, segH = nullHandle),
  --   one space for the original heap (isLargeNode = FALSE, segH = nullHandle), and
  --   one space for each extension segment (isLargeNode = FALSE, segH~ = nullHandle).
  -- Things are arranged so P may delete the space:
  --   next of a large node is saved, and
  --   the space for the original heap is saved for the last call on P.
  BEGIN
  hasSwapUnits: BOOLEAN = h.swapUnit ~= defaultSwapUnit;
  largeNode, nextLargeNode: LONG POINTER TO LargeNodeHeader;
  firstSeg, seg: LONG POINTER;
  segH, nextSegH: Zone.SegmentHandle;
  -- First enumerate the large node spaces.
  FOR largeNode ← h.largeNodes, nextLargeNode WHILE largeNode ~= NIL DO
    nextLargeNode ← largeNode.next;
    P[SpaceFromLongPointer[largeNode, hasSwapUnits], TRUE, Zone.nullSegment];
    ENDLOOP;
  -- Now enumerate the segments, visiting the original segment last.
  [storage: firstSeg, next: nextSegH] ← Zone.GetAttributes[h.zH];
  -- Visit all those segments added to the zone.
  WHILE nextSegH ~= Zone.nullSegment DO
    segH ← nextSegH;
    [storage: seg, next: nextSegH] ← Zone.GetSegmentAttributes[h.zH, segH];
    P[SpaceFromLongPointer[seg, hasSwapUnits], FALSE, segH];
    ENDLOOP;
  -- Finally visit the segment from the main zone.
  P[SpaceFromLongPointer[firstSeg, hasSwapUnits], FALSE, Zone.nullSegment];
  END;

FreeLargeOrRegularNode: ENTRY PROCEDURE [h: Handle, p: LONG POINTER] =
  -- Free the space created by MakeLargeNode,
  --   or if this is a false alarm just free a (large) normal node.
  BEGIN
  prev: LONG POINTER TO LONG POINTER TO LargeNodeHeader ← @h.largeNodes;
  largeNode: LONG POINTER TO LargeNodeHeader;
  DO
    largeNode ← prev↑;
    IF largeNode = NIL THEN GOTO Regular;
    IF @largeNode.node = p THEN GOTO Large;
    prev ← @largeNode.next;
    REPEAT
      Large =>
        BEGIN
        prev↑ ← largeNode.next;
        FreeSpace[p, h.swapUnit ~= defaultSwapUnit];
        END;
      Regular =>
        SELECT Zone.FreeNode[zH: h.zH, p: p] FROM
```

```
        okay => NULL;
        invalidNode => ERROR Error[invalidNode];
        ENDCASE => ERROR Error[invalidHeap];
    ENDLOOP;
  END;

  FreeSpace: PROCEDURE [p: LONG POINTER, hasSwapUnits: BOOLEAN] =
    -- Delete the mapped space containing the address p.
    BEGIN Space.Delete[SpaceFromLongPointer[p, hasSwapUnits]]; END;

  MakeLargeNode: ENTRY PROCEDURE [h: Handle, n: NWords] RETURNS [LONG POINTER] =
    -- Make a space to hold a large node, and add it to h's largeNodes list.
    BEGIN
    largeNode: LONG POINTER TO LargeNodeHeader = MakeSpace[
      h.parent, PagesForWords[n + SIZE[LargeNodeHeader]], h.swapUnit, h.resident !
      Space.InsufficientSpace => GOTO InsufficientSpace;
      Space.Error => GOTO UnsuitableParent];
    largeNode↑ ←
      [next: h.largeNodes,
        nodeHeader: [length: n + nodeOverhead, extension: inuse[]], node: NULL];
    h.largeNodes ← largeNode;
    RETURN[@largeNode.node]
    EXITS
      InsufficientSpace => RETURN WITH ERROR Error[insufficientSpace];
      UnsuitableParent => RETURN WITH ERROR Error[unsuitableParent];
    END;

  MakeSpace: PROCEDURE [
    parent: Space.Handle, pages, swapUnit: Environment.PageCount,
    resident: BOOLEAN] RETURNS [LONG POINTER] =
    -- Make a data space, and perhaps (one level of) swap units.
    BEGIN
    sH: Space.Handle = Space.Create[size: pages, parent: parent];
    IF swapUnit ~= defaultSwapUnit THEN
      Space.CreateUniformSwapUnits[size: swapUnit, parent: sH];
    Space.Map[sH];
    IF resident THEN SpecialSpace.MakeResident[sH];
    RETURN[Space.LongPointer[sH]]
    END;

  Narrow: PROCEDURE [p: LONG POINTER] RETURNS [POINTER] =
    -- Convert a long pointer assumed to point into this mds into a short pointer.
    BEGIN RETURN[LOOPHOLE[Inline.LowHalf[p - pMDS], POINTER]] END;

  PagesForNewSegment: PROCEDURE [n: NWords] RETURNS [Space.PageCount] =
    BEGIN RETURN[PagesForWords[zoneOverhead + nodeOverhead + n]] END;

  PagesForWords: PROCEDURE [n: NWords] RETURNS [Environment.PageCount] =
    -- Calculate the number of whole pages to hold n words.
    INLINE
    BEGIN RETURN[(n + Environment.wordsPerPage - 1)/Environment.wordsPerPage] END;

  SpaceFromLongPointer: PROCEDURE [p: LONG POINTER, hasSwapUnits: BOOLEAN]
    RETURNS [sh: Space.Handle] =
    -- Find the mapped space containing the address p.
    BEGIN
    space: Space.Handle = Space.GetHandle[Space.PageFromLongPointer[p]];
    parent: Space.Handle = Space.GetAttributes[space].parent;
    RETURN[IF hasSwapUnits THEN parent ELSE space]
```



```
    END;

  END.
```

May 23, 1980 1:45 PM McJones Create file
June 10, 1980 4:32 PM McJones Convert back to Amargosa
June 11, 1980 2:54 PM McJones Reconvert to Mesa 6
June 30, 1980 4:38 PM McJones Identify systemHeap with SystemMDSHeap if UtilityPilo
***t*
July 8, 1980 5:06 PM McJones Don't catch Space.InsufficientSpace in MakeSpace
July 19, 1980 3:40 PM McJones New ZONE representation; fix bugs in large node detec
***tion and shortening large node pointer in MakeMDSNode; store large node header right in large*
***node*
July 24, 1980 10:07 AM McJones Identify systemHeap with SystemMDSHeap if UtilityPilo
***t (again!)*
July 26, 1980 3:24 PM Forrest If UtilityPilot treat large MDS nodes same as if small
August 4, 1980 6:17 PM McJones Large nodes and expansions of resident heap must be
***made resident; EnumerateSpaces must be prepared for deletion of large node space; repackag*
***e Expand for export*
August 5, 1980 1:54 PM McJones systemZONE = >SystemZone; delete {Make/Free}[MD
***S]String procedures made inline in Heap*

```
-- Swapper>ResidentHeapImpl.mesa (last edited by Knutsen on April 14, 1980 9:12 AM)

DIRECTORY
  Environment: FROM "Environment" USING [Base, Long, wordsPerPage],
  Inline: FROM "Inline" USING [BITAND],
  MiscPrograms: FROM "MiscPrograms",
  ResidentHeap: FROM "ResidentHeap" USING [first64K],
  ResidentMemory: FROM "ResidentMemory" USING [Allocate, Free, Location],
  RuntimeInternal: FROM "RuntimeInternal" USING [WorryCallDebugger],
  Zone: FROM "Zone" USING [
    AddSegment, Alignment, BlockSize, Create, FreeNode, Handle, MakeNode,
    SplitNode, Status],
  ZoneInternal: FROM "ZoneInternal" USING [NodeHeader];

ResidentHeapImpl: MONITOR
  IMPORTS Inline, ResidentMemory, RuntimeInternal, Zone
  EXPORTS ResidentHeap, MiscPrograms =
  BEGIN
  --      Global types
  InuseNodePointer: TYPE = POINTER TO inuse ZoneInternal.NodeHeader;
  --      Global constants for the resident heap (these should be adjusted to best reflect the de
**mands of Pilot)
  --Warning: The next two constants must be one as some of the clients of this module on some pr
**ocessors can not have an object crossing a page boundary. (For instance, IOCB's can not cros
**s a page boundary on the Dandelion.)
  initialHeapSize: CARDINAL ← 1;
  --pages (this is what pilot currently needs initially)
  heapSizeIncrement: CARDINAL ← 1; --pages
  maxExtraSegments: CARDINAL ← 18; --increments; testPackage>rs232C needs these
  inuseNodeHeaderSize: CARDINAL = SIZE[inuse ZoneInternal.NodeHeader];
  largeObjectThreshold: Zone.BlockSize = Environment.wordsPerPage/2;
  -- any objects this size or larger will be allocated as separate pages and not in residentZone.
  largeNodeThreshold: Zone.BlockSize = largeObjectThreshold + inuseNodeHeaderSize;
  --      Global variables for the resident heap
  residentZone: PUBLIC Zone.Handle;
  status: Zone.Status;
  base: Environment.Base = ResidentHeap.first64K; --to save typing!
  numberOfExtraSegments: CARDINAL ← 0;
  --      Statistical counters
  allocates, frees, splits: CARDINAL ← 0;
  alignmentDistribution: ARRAY Zone.Alignment OF CARDINAL ← ALL[0];
  sizeDistribution: ARRAY [0..nSizes] OF CARDINAL ← ALL[0];
  nSizes: CARDINAL = 11;
  sizes: ARRAY [0..nSizes) OF Zone.BlockSize =
    [4, 8, 12, 16, 24, 32, 48, 64, 128, 256, 512];

  InitializeResidentHeap: PUBLIC PROCEDURE =
    BEGIN
    [residentZone, status] ← Zone.Create[
      storage: ResidentMemory.Allocate[first64K, initialHeapSize],
      length: Environment.wordsPerPage*initialHeapSize, zoneBase: base];
    IF status ~= okay THEN
      DO RuntimeInternal.WorryCallDebugger["Can't init resident heap"] ENDLOOP;
    END;
  --      Implementation of the heap operations

  FreeNode: PUBLIC ENTRY PROCEDURE [p: Environment.Base RELATIVE POINTER]
    RETURNS [s: Zone.Status] =
```



```
    BEGIN
    nodeP: InuseNodePointer = LOOPHOLE[RelativePointerToPointer[
      p - inuseNodeHeaderSize]];
    frees ← frees + 1;
    IF nodeP.length >= largeNodeThreshold THEN
      BEGIN
      -- Assumption: alignments are at no greater than single page boundaries. Otherwise, Words
**ToPages is not returning the actual number of pages allocated
      ResidentMemory.Free[first64K, WordsToPages[nodeP.length], @base[p]];
      RETURN[okay];
      END
    ELSE RETURN[Zone.FreeNode[zH: residentZone, p: @base[p]]];
    END;

  Increment: PRIVATE ENTRY PROCEDURE [count: POINTER TO CARDINAL] = INLINE
    BEGIN count↑ ← count↑ + 1; END;

  LargeNode: PROCEDURE [node: Environment.Base RELATIVE POINTER]
    RETURNS [BOOLEAN] =
    BEGIN
    nodeP: InuseNodePointer = LOOPHOLE[RelativePointerToPointer[
      node - inuseNodeHeaderSize]];
    RETURN[nodeP.length > largeNodeThreshold];
    END;

  MakeNode: PUBLIC ENTRY PROCEDURE [n: Zone.BlockSize, alignment: Zone.Alignment]
    RETURNS [node: Environment.Base RELATIVE POINTER, s: Zone.Status] =
    BEGIN
    UpdateCounts: PROCEDURE =
      BEGIN
      i: [0..nSizes);
      allocates ← allocates + 1;
      alignmentDistribution[alignment] ← alignmentDistribution[alignment] + 1;
      FOR i IN [0..nSizes) DO
        IF n < sizes[i] THEN
          BEGIN sizeDistribution[i] ← sizeDistribution[i] + 1; EXIT; END;
        REPEAT
          FINISHED => sizeDistribution[nSizes] ← sizeDistribution[nSizes] + 1;
        ENDLOOP;
      END;
    IF n >= largeObjectThreshold THEN
      BEGIN
      alignmentModulii: ARRAY Zone.Alignment OF CARDINAL = [1, 2, 4, 8, 16];
      -- entry for alignment "ai" is i
      masks: ARRAY Zone.Alignment OF CARDINAL =
        [177777B, 177776B, 177774B, 177770B, 177760B];
      -- entry for alignment "ai" is 177777B left shifted by log2i-1. Thus i must be a power of 2.
      alignmentModulus: CARDINAL; -- Cardinal'ized version of alignment
      mask: CARDINAL; -- used to turn a pointer into a pointer to aligned storage
      nodeOffset: CARDINAL; -- Offset of the allocated node
      size: CARDINAL; -- size of the allcoated node (in words)
      longNodeP: Environment.Long;
      nodeP: InuseNodePointer;
      alignmentModulus ← alignmentModulii[alignment];
      mask ← masks[alignment];
      nodeOffset ←
        Inline.BITAND[inuseNodeHeaderSize + alignmentModulus - 1, mask] -
          inuseNodeHeaderSize;
      -- align the node so that the data area is properly aligned as per alignment
```

```
    size ← nodeOffset + inuseNodeHeaderSize + n;
    longNodeP.lp ← ResidentMemory.Allocate[first64K, WordsToPages[size]];
    nodeP ← LOOPHOLE[longNodeP.lowbits + nodeOffset];
    nodeP↑ ← ZoneInternal.NodeHeader[length: size, extension: inuse[]];
    UpdateCounts[];
    RETURN[
      LOOPHOLE[LOOPHOLE[LOOPHOLE[LONG[nodeP + inuseNodeHeaderSize], LONG
                    POINTER] - base, Environment.Long].lowbits], okay];
    END
  ELSE -- Else it is a small object so use residentZone
    BEGIN
    [node, s] ← Zone.MakeNode[zH: residentZone, n: n, alignment: alignment];
    IF s = noRoomInZone THEN
      IF numberOfExtraSegments < maxExtraSegments THEN
        BEGIN
        status ← Zone.AddSegment[
          zH: residentZone,
          storage: ResidentMemory.Allocate[first64K, heapSizeIncrement],
          length: Environment.wordsPerPage*heapSizeIncrement].s;
        IF status = okay THEN numberOfExtraSegments ← numberOfExtraSegments + 1
        ELSE
          RuntimeInternal.WorryCallDebugger["Resident heap extension error."];
        [node, s] ← Zone.MakeNode[zH: residentZone, n: n, alignment: alignment];
        IF s = noRoomInZone THEN
          RuntimeInternal.WorryCallDebugger["Node too large for Resident Heap"];
        END;
    IF s = okay THEN UpdateCounts[];
    END;
  END;

NodeSize: PUBLIC ENTRY PROCEDURE [p: Environment.Base RELATIVE POINTER]
  RETURNS [n: Zone.BlockSize] =
  BEGIN
  RETURN[
    LOOPHOLE[RelativePointerToPointer[p - inuseNodeHeaderSize],
      InuseNodePointer].length]
  END;

RelativePointerToPointer: PRIVATE PROCEDURE [
  p: Environment.Base RELATIVE POINTER] RETURNS [POINTER] =
  BEGIN
  longNodeP: Environment.Long;
  longNodeP.lp ← @base[p];
  RETURN[LOOPHOLE[longNodeP.low]];
  END;

SplitNode: PUBLIC PROCEDURE [
  p: Environment.Base RELATIVE POINTER, n: Zone.BlockSize]
  RETURNS [s: Zone.Status] =
  -- Splitting of large object nodes is not supported due the percieved needs of the clients of larg
**e nodes
  BEGIN
  Increment[@splits];
  IF LargeNode[p] THEN RETURN[okay]
  ELSE RETURN[Zone.SplitNode[zH: residentZone, p: @base[p], n: n]];
  END;

WordsToPages: PRIVATE PROCEDURE [n: CARDINAL] RETURNS [CARDINAL] =
  BEGIN
```



```
  RETURN[(n + Environment.wordsPerPage - 1)/Environment.wordsPerPage];
  END;

END.
```

LOG

Time: April 30, 1979 9:18 PM By: Lauer Action: Created file

Time: June 2, 1979 10:47 PM By: Lauer Action: Replaced the stub in this file with the real imple
***mentation of the resident heap*

Time: July 18, 1979 2:57 PM By: Knutsen Action: Merged contents of obsolete ZoneEx
***tension into Zone and Environment.*

Time: August 17, 1979 4:28 PM By: Lauer Action: Changed initialHeapSize, heapSizeIncrement,
***maxExtraSegments to variables; improved statistics gathering in sizeDistribution*

Time: September 13, 1979 10:48 AM By: Schwartz Action: Allowed nodes slightly lar
***ger than one page (for Gateway communication with Xerox 850), and corrected diagnosis of att*
***empts to allocate nodes larger than maxNodeSize.*

Time: November 16, 1979 9:56 AM By: Forrest Action: Fixed max number of incr
***ement's logic. Made MakeNode an entry procedure, since the test for extending the node has t*
***o be "atomic".*

Time: November 20, 1979 6:54 PM By: Forrest Action: Raised number of increm
***ents to 5, since this is what TestPackage>Exercise RS232C needs to run. Also incresed initial al*
***location to 3*

Time: April 10, 1980 2:55 PM By: Luniewski Action: Allocate large objects in the first 64K
***but not in the actual resident zone in order to avoid fragmentation problems in the zone. Also,*
***modify so that small objects do not cross page boundaries.*

Time: April 14, 1980 9:12 AM By: Knutsen Action: Now STARTed by InitializeResidentH
***eap.*

```
-- StreamImpl.mesa (last edited by: Forrest on: May 23, 1980  12:12 PM)

DIRECTORY
  BitBlt: FROM "BitBlt" USING [AlignedBBTable, BITBLT, BBptr, BBTableSpace],
  ByteBlt: FROM "ByteBlt",
  Environment: FROM "Environment" USING [
    bitsPerWord, bitsPerByte, Block, Byte, bytesPerWord],
  Inline: FROM "Inline" USING [LongCOPY],
  MiscPrograms: FROM "MiscPrograms",
  PrincOps: FROM "PrincOps" USING [UnboundLink],
  Stream: FROM "Stream",
  SystemInternal: FROM "SystemInternal" USING [Unimplemented];

StreamImpl: PROGRAM
  IMPORTS Inline, BitBlt, SystemInternal EXPORTS ByteBlt, MiscPrograms, Stream =
  BEGIN OPEN Stream;
  --MiscPrograms.--
  InitializeStream: PUBLIC PROCEDURE =
    BEGIN -- may be something here someday -- END;

  EndOfStream: PUBLIC SIGNAL [nextIndex: CARDINAL] = CODE;
  LongBlock: PUBLIC SIGNAL [nextIndex: CARDINAL] = CODE;
  ShortBlock: PUBLIC ERROR = CODE;
  SSTChange: PUBLIC SIGNAL [sst: SubSequenceType, nextIndex: CARDINAL] = CODE;
  TimeOut: PUBLIC SIGNAL [nextIndex: CARDINAL] = CODE;
  LeftAndRight: TYPE = MACHINE DEPENDENT RECORD [left, right: Environment.Byte];
  --
  -- Stream implementation (defaults)
  defaultObject: PUBLIC Object ←
    [options: defaultInputOptions, getByte: DefaultGetByte,
      putByte: DefaultPutByte, getWord: DefaultGetWord, putWord: DefaultPutWord,
      get: DefaultGet, put: DefaultPut, setSST: LOOPHOLE[PrincOps.UnboundLink],
      sendAttention: LOOPHOLE[PrincOps.UnboundLink],
      waitAttention: LOOPHOLE[PrincOps.UnboundLink],
      delete: LOOPHOLE[PrincOps.UnboundLink]];
  DefaultGetByte: PROCEDURE [sH: Handle] RETURNS [byte: Byte] =
    -- Get one byte, using GetBlock on a one-byte block
    BEGIN
    array: PACKED ARRAY [0..1] OF Byte;
    options: InputOptions = [FALSE, FALSE, FALSE, TRUE, TRUE];
    IF sH.get = DefaultGet THEN ERROR SystemInternal.Unimplemented;
    [] ← sH.get[sH, Block[@array, 1, 2], options];
    RETURN[array[1]] -- right-justify the byte

    END;

  DefaultGetWord: PROCEDURE [sH: Handle] RETURNS [word: Word] =
    -- Get one word using GetBlock on a two-byte block
    BEGIN OPEN w: LOOPHOLE[word, LeftAndRight];
    options: InputOptions = [FALSE, FALSE, FALSE, TRUE, TRUE];
    SELECT TRUE FROM
      sH.getByte ~= DefaultGetByte =>
        BEGIN w.left ← sH.getByte[sH]; w.right ← sH.getByte[sH]; END;
      sH.get ~= DefaultGet => [] ← sH.get[sH, Block[@word, 0, 2], options];
      ENDCASE => ERROR SystemInternal.Unimplemented;
    END;

  DefaultGet: PROCEDURE [sH: Handle, block: Block, options: InputOptions]
```



```
    RETURNS [
      bytesTransferred: CARDINAL, why: CompletionCode, sst: SubSequenceType] =
    -- Get block, using GetByte repeatedly
    BEGIN
    i: CARDINAL;
    sstNew: SubSequenceType;
    IF sH.getByte = DefaultGetByte OR options.terminateOnEndPhysicalRecord OR
      options.signalLongBlock OR options.signalShortBlock THEN
      ERROR SystemInternal.Unimplemented;
    why ← normal;
    i ← block.startIndex;
    WHILE i < block.stopIndexPlusOne DO
      LOOPHOLE[block.blockPointer, LONG POINTER TO PACKED ARRAY [0..0) OF Byte][i]
        ← sH.getByte[
        sH !
        SSTChange =>
          IF ~options.signalSSTChange THEN
            BEGIN sstNew ← sst; GO TO PostSSTChange END
          ELSE BEGIN SIGNAL SSTChange[sst: sst, nextIndex: i]; RESUME END;
        EndOfStream =>
          IF ~options.signalEndOfStream THEN GO TO PostEndOfStream
          ELSE
            BEGIN
            SIGNAL EndOfStream[nextIndex: i];
            RESUME -- why is this useful? --
            END];
      i ← i + 1;
      REPEAT
        PostSSTChange => why ← sstChange;
        PostEndOfStream => why ← endOfStream;
      ENDLOOP;
    bytesTransferred ← i - block.startIndex;
    sst ← sstNew
    END;

  DefaultPutByte: PROCEDURE [sH: Handle, byte: Byte] =
    -- Put one byte, using PutBlock on a one-byte block
    BEGIN
    array: PACKED ARRAY [0..1] OF Byte ← [0, byte];
    IF sH.put = DefaultPut THEN ERROR SystemInternal.Unimplemented;
    sH.put[sH, Block[@array, 1, 2], FALSE];
    END;

  DefaultPutWord: PROCEDURE [sH: Handle, word: Word] =
    -- Put one word, using PutBlock on a one-byte block
    BEGIN OPEN w: LOOPHOLE[word, LeftAndRight];
    SELECT TRUE FROM
      sH.putByte ~= DefaultPutByte =>
        BEGIN sH.putByte[sH, w.left]; sH.putByte[sH, w.right]; END;
      sH.put ~= DefaultPut => sH.put[sH, Block[@word, 0, 2], FALSE];
      ENDCASE => ERROR SystemInternal.Unimplemented;
    END;

  DefaultPut: PROCEDURE [sH: Handle, block: Block, endPhysicalRecord: BOOLEAN] =
    -- Put block, using PutByte repeatedly
    BEGIN
    i: CARDINAL;
    IF sH.putByte = DefaultPutByte OR endPhysicalRecord THEN
      ERROR SystemInternal.Unimplemented;
```

```
    FOR i IN [block.startIndex..block.stopIndexPlusOne) DO
      sH.putByte[
        sH, LOOPHOLE[block.blockPointer, LONG POINTER TO PACKED ARRAY [0..0) OF
        Byte][i]];
      ENDLOOP;
    END;
  --
  -- ByteBlt implementation

  useBitBlt: BOOLEAN ← FALSE;
  ByteBlt: PUBLIC PROCEDURE [to, from: Environment.Block]
    RETURNS [nBytes: CARDINAL] =
    -- Byte-boundary block transfer
    BEGIN
    -- NB: to&from are RECORDs, not POINTERs to RECORDs, so we can update them
    toBytes, fromBytes: LONG POINTER TO PACKED ARRAY [0..0) OF [0..377B);
    moved: CARDINAL ← 0;
    -- This check is necessary since subtracting CARDINALs gives big numbers
    IF to.startIndex > to.stopIndexPlusOne OR from.startIndex >
      from.stopIndexPlusOne THEN ERROR StartIndexGreaterThanStopIndexPlusOne;
    IF
      (nBytes ← MIN[
        to.stopIndexPlusOne - to.startIndex,
          from.stopIndexPlusOne - from.startIndex]) = 0 THEN RETURN;
    toBytes ← to.blockPointer;
    fromBytes ← from.blockPointer;
    -- Move the first odd byte (if any) to be sure that to is word aligned
    IF (to.startIndex MOD 2) # 0 THEN
      BEGIN
      toBytes[to.startIndex] ← fromBytes[from.startIndex];
      moved ← 1;
      to.startIndex ← to.startIndex + 1;
      from.startIndex ← from.startIndex + 1;
      END;
    IF (from.startIndex MOD 2) = 0 THEN
      -- Fast case: both are word aligned
      BEGIN
      words: CARDINAL = (nBytes - moved)/2;
      Inline.LongCOPY[
        to: toBytes + to.startIndex/2, from: fromBytes + from.startIndex/2,
        nwords: words];
      IF (moved + 2*words) # nBytes THEN
        -- Move the one and only remaining byte
        toBytes[to.startIndex + 2*words] ← fromBytes[from.startIndex + 2*words];
      END
      -- Slow case: have to ripple things

    ELSE
      IF ~useBitBlt THEN
        BEGIN
        i: CARDINAL;
        count: CARDINAL = nBytes - moved;
        FOR i IN [0..count) DO
          toBytes[to.startIndex + i] ← fromBytes[from.startIndex + i]; ENDLOOP;
        END
      ELSE
        -- BitBlt is not interruptable except at the end of each scan line, so we break things up into
**chunks in order to maintain reasonable interrupt latency for the IO devices.  It takes about 200m
**icrosec to move 50 bytes with the display off.
        BEGIN
```



```
        bba: BitBlt.BBTableSpace;
        bbt: BitBlt.BBptr = BitBlt.AlignedBBTable[@bba];
        lineWidth: CARDINAL = 16;
        -- words per scan line: controls interrupt latency
        bitsPerLine: CARDINAL = lineWidth*Environment.bitsPerWord;
        bytesPerLine: CARDINAL = lineWidth*Environment.bytesPerWord;
        lines: CARDINAL = (nBytes - moved)/bytesPerLine;
        -- bytes left to move with first BitBlt
        tail: CARDINAL = (nBytes - moved) MOD bytesPerLine;
        -- bytes left to move with second BitBlt
        bbt↑ ←
          [dst: [word: toBytes + to.startIndex/2, bit: 0], dstBpl: bitsPerLine,
            src: [word: fromBytes + from.startIndex/2, bit: 8],
            srcDesc: [srcBpl[bitsPerLine]], width: bitsPerLine, height: lines,
            flags:
            [direction: forward, disjoint: TRUE, disjointItems: TRUE, gray: FALSE,
              srcFunc: null, dstFunc: null]];
        -- This BitBlt moves a rectangle that is lineWidth words wide by as many lines high as will fit
**.  NB: It cheats and actually reads a byte from beyond the edge of the rectangle.  This is not reall-
**y legal, but works out OK for any reasonable implementation of BitBlt.
        IF lines # 0 THEN BitBlt.BITBLT[bbt];
        -- update the pointers to reflect the work done, and then move one line that is less than line
**Width words wide.
        bbt.dst.word ← bbt.dst.word + lines*lineWidth;
        bbt.src.word ← bbt.src.word + lines*lineWidth;
        bbt.width ← Environment.bitsPerByte*tail;
        bbt.height ← 1;
        IF tail # 0 THEN BitBlt.BITBLT[bbt];
        END;
    END;

  StartIndexGreaterThanStopIndexPlusOne: PUBLIC ERROR = CODE;
  END.
LOG
Time: April 11, 1980  6:37 PM    By: Forrest          Action: Trimmed log to Amargosa: Converte
**d ByteBlt to use PrincOps BitBlt
Time: April 14, 1980  8:39 AM    By: Knutsen          Action: Module STARTed by InitializeStream
**.
Time: April 16, 1980  9:07 PM    By: Forrest          Action: Stopped using BitBlt until it is debug
**ged.
Time: April 28, 1980  10:12 AM   By: Forrest          Action: ControlDefs => PrincOps.
```

```
-- Misc>UtilitiesImpl.mesa (last edited by Forrest on July 18, 1980 4:05 PM PM)

DIRECTORY
  Inline USING [LongCOPY],
  MiscPrograms USING [],
  Utilities USING [];

UtilitiesImpl: PROGRAM IMPORTS Inline EXPORTS MiscPrograms, Utilities =
  BEGIN
  InitializeUtilities: PUBLIC PROC = { -- may be something here someday --};
  -- Move the contents of the size words starting at pSource to the size words starting at pSink
  -- Unlike Inline.LongCOPY, overlapping blocks do not cause replication of source words.

  LongMove: PUBLIC PROC [
    pSource: LONG POINTER, size: CARDINAL, pSink: LONG POINTER] =
    BEGIN
    LP: PROC [p: LONG POINTER] RETURNS [LONG ORDERED POINTER] = INLINE {
      RETURN[LOOPHOLE[p]]};
    i: CARDINAL;
    IF LP[pSink] IN [LP[pSource]..LP[pSource] + size) THEN
      FOR i DECREASING IN [0..size) DO (pSink + i)↑ ← (pSource + i)↑ ENDLOOP
    ELSE Inline.LongCOPY[from: pSource, nwords: size, to: pSink]
    END;

  END.
LOG
Time: April 14, 1980 8:39 AM   By: Knutsen   Action: Added InitializeUtilities.
Time: July 17, 1980 6:41 PM    By: Forrest   Action: Deleted Page< = >LongPointer ops,
**which are now INLINE
```

```
-- Misc>ZoneImpl.mesa (last edited by Forrest on September 20, 1980 8:07 PM)
-- A set of procedures to manage allocation within a zone. Coalescing of free nodes occurs during
**allocation; all free nodes following a candidate node are merged before any space is allocated.
**The logic is derived from a BCPL program by E. M. McCreight and was suggested by an exercis
**e in Knuth Volume I, p. 453 # 19
--          This version was adapted for Pilot from the Free Storage Package of the Mesa 4.0 Syste
**m. The principle differences are: (1) the zone is managed in terms of 16-bit RELATIVE POINTER
**s relative to a client supplied BASE; (2) no SIGNALs are raised, since the implementation is use
**d by resident parts of Pilot which cannot invoke the Signaller, and (3) the mechanism for adding
**space to a zone is changed.

DIRECTORY
  Environment USING [Base],
  Inline USING [BITAND, BITSHIFT, LongNumber],
  Process USING [InitializeMonitor],
  MiscPrograms USING [],
  Zone USING [Alignment, BlockSize, Handle, SegmentHandle, Status],
  ZoneInternal USING [
    Base, FreeNodePointer, InuseNodePointer, NodeHeader, NodePointer, RPtr,
    SegmentHeader, SegmentPointer, ZoneHeader, ZonePointer];

ZoneImpl: MONITOR LOCKS LOOPHOLE[zH, ZoneInternal.ZonePointer] USING zH:
  Zone.Handle IMPORTS Inline, Process EXPORTS Zone, MiscPrograms =
  BEGIN

  usedNodeSize: Zone.BlockSize = SIZE[inuse ZoneInternal.NodeHeader];
  freeNodeSize: Zone.BlockSize = SIZE[free ZoneInternal.NodeHeader];
  zoneHeaderSize: Zone.BlockSize = SIZE[ZoneInternal.ZoneHeader];
  segmentHeaderSize: Zone.BlockSize = SIZE[ZoneInternal.SegmentHeader];
  zoneOverhead: CARDINAL =
    SIZE[ZoneInternal.ZoneHeader] + SIZE[inuse ZoneInternal.NodeHeader];
  nodeOverhead: CARDINAL = SIZE[inuse ZoneInternal.NodeHeader];

  -- NOTE: A zone whose largest possible node is N words, must have
  -- N + zoneOverhead + nodeOverhead words of storage
  nilSeg: ZoneInternal.SegmentPointer = LOOPHOLE[NIL];

  -- Exported (READONLY) variables
  minimumNodeSize: PUBLIC Zone.BlockSize ← freeNodeSize;
  nil: PUBLIC Environment.Base RELATIVE POINTER ← LOOPHOLE[0];
  nullSegment: PUBLIC Zone.SegmentHandle ← LOOPHOLE[nilSeg];

  -- Module initialization:
  --MiscPrograms.--
  InitializeZone: PUBLIC PROC = { --may be something here eventually--};

  -- procedures

  -- Adds a segment to the zone zH. The storage must be capable of being addressed with 16-bit p
**ointers relative to the zoneBase of the zone.

  AddSegment: PUBLIC ENTRY PROC [
    zH: Zone.Handle, storage: LONG POINTER, length: Zone.BlockSize]
    RETURNS [sH: Zone.SegmentHandle, s: Zone.Status] =
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    zb: ZoneInternal.Base = z.zoneBase;
    st: LONG ORDERED POINTER = LOOPHOLE[storage];
    sp: ZoneInternal.SegmentPointer = Relative[zb, st];
```



```
    fn: ZoneInternal.FreeNodePointer;
    an: ZoneInternal.InuseNodePointer;
    IF LOOPHOLE[length, CARDINAL] > LAST[Zone.BlockSize] THEN
      RETURN[sH: nullSegment, s: storageOutOfRange];
    IF zb > st OR st + length - 1 > zb + LAST[CARDINAL] THEN
      RETURN[sH: nullSegment, s: storageOutOfRange];
    IF length < segmentHeaderSize + MAX[freeNodeSize, z.threshold] + usedNodeSize
      THEN RETURN[sH: nullSegment, s: segmentTooSmall];
    IF length > LAST[Zone.BlockSize] THEN
      RETURN[sH: nullSegment, s: storageOutOfRange];
    IF (s ← ValidateZone[zH]) ~= okay THEN RETURN[sH: nullSegment, s: s];
    -- set up the bulk of the segment as a large free block.
    fn ← LOOPHOLE[sp + segmentHeaderSize, ZoneInternal.FreeNodePointer];
    zb[fn] ← ZoneInternal.NodeHeader[
      length: length - (segmentHeaderSize + usedNodeSize),
      extension: free[fwdp: z.freeList, backp: z.node.backp]];
    z.node.backp ← zb[zb[fn].backp].fwdp ← fn;
    -- set up allocated node (smallest possible) at end of block.
    an ← LOOPHOLE[sp + (length - usedNodeSize), ZoneInternal.InuseNodePointer];
    zb[an] ← ZoneInternal.NodeHeader[length: usedNodeSize, extension: inuse[]];
    -- set up the segment header and link it into the chain of segments
    zb[sp] ← ZoneInternal.SegmentHeader[
      length: length, nextSegment: z.nextSegment];
    z.nextSegment ← sp;
    RETURN[sH: LOOPHOLE[sp, Zone.SegmentHandle], s: okay];
    END;

  -- Checks to see that the node is properly part of this zone and is correctly linked to the free list

  CheckNode: PROC [zH: Zone.Handle, node: ZoneInternal.NodePointer]
    RETURNS [s: Zone.Status] =
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    zb: ZoneInternal.Base = z.zoneBase;
    --       First check to see that node lies within this zone.
    BEGIN
    sp: ZoneInternal.SegmentPointer;
    rz: ZoneInternal.RPtr = Relative[zb, LOOPHOLE[zH, ZoneInternal.ZonePointer]];
    --get the zone pointer relative to the zone base
    endNode: ZoneInternal.RPtr = node + zb[node].length;
    --the address of the end of the node
    IF node > rz AND endNode < rz + z.length THEN GOTO okay;
    --is node in primary part of zone?
    FOR sp ← z.nextSegment, zb[sp].nextSegment WHILE sp ~= nilSeg DO
      IF node > LOOPHOLE[sp, ZoneInternal.RPtr] AND endNode < sp + zb[sp].length
        THEN GOTO okay; --is node in this segment of zone?

      ENDLOOP;
    RETURN[invalidNode]; --node is not in any segment of the zone

    EXITS okay => NULL;
    END;
    --       Now check that node is properly linked on the free list
    IF zb[node].state = free THEN
      DO
        WITH zb[node] SELECT FROM
          inuse => IF length = usedNodeSize THEN EXIT; -- end of zone

          free =>
```

```
        BEGIN
        IF zb[fwdp].backp # node OR zb[backp].fwdp # node THEN
          RETURN[invalidNode];
        IF length = 0 AND node # z.freeList THEN RETURN[invalidNode];
        END;
      ENDCASE;
    node ← node + zb[node].length;
    IF zb[node].state # inuse THEN EXIT;
    ENDLOOP;
  RETURN[okay]
  END;

CheckZone: PROC [zH: Zone.Handle] RETURNS [s: Zone.Status] =
  -- Checks that the zone is well-formed and that its free list is intact
  BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
  zb: ZoneInternal.Base = z.zoneBase;
  node: ZoneInternal.FreeNodePointer;
  count: INTEGER;
  IF (s ← ValidateZone[zH]) ~= okay THEN RETURN[invalidZone];
  count ← (LAST[Zone.BlockSize] - FIRST[Zone.BlockSize])/freeNodeSize + 1;
  node ← z.freeList;
  DO
    IF (s ← CheckNode[zH, node]) ~= okay THEN RETURN[invalidNode];
    IF (count ← count - 1) < 0 THEN RETURN[nodeLoop];
    IF (node ← zb[node].fwdp) = z.freeList THEN EXIT;
    ENDLOOP;
  RETURN[okay]
  END;

-- Makes a zone out of the storage provided by the caller. The caller also provides a LONG BASE
**POINTER to which all addressing within the zone will be RELATIVE. This LONG BASE POINTER
**must cover all of the storage.

Create: PUBLIC PROC [
  storage: LONG POINTER, length: Zone.BlockSize, zoneBase: Environment.Base,
  threshold: Zone.BlockSize, checking: BOOLEAN]
  RETURNS [zH: Zone.Handle, s: Zone.Status] =
  BEGIN
  fn: ZoneInternal.FreeNodePointer;
  an: ZoneInternal.InuseNodePointer;
  z: ZoneInternal.ZonePointer = LOOPHOLE[storage];
  zb: ZoneInternal.Base = LOOPHOLE[zoneBase];
  rp: ZoneInternal.RPtr = Relative[zb, z];
  freeList: ZoneInternal.FreeNodePointer = Relative[zb, @z.node];
  IF LOOPHOLE[length, CARDINAL] > LAST[Zone.BlockSize] THEN
    RETURN[zH: LOOPHOLE[z, Zone.Handle], s: storageOutOfRange];
  IF zb > z OR z + length - 1 > zb + LAST[CARDINAL] THEN
    RETURN[zH: LOOPHOLE[z, Zone.Handle], s: storageOutOfRange];
  IF length < zoneHeaderSize + MAX[freeNodeSize, threshold] + usedNodeSize THEN
    RETURN[zH: LOOPHOLE[z, Zone.Handle], s: zoneTooSmall];
  IF length > LAST[Zone.BlockSize] THEN
    RETURN[zH: LOOPHOLE[z, Zone.Handle], s: storageOutOfRange];
  -- set up the bulk of the zone as a large free block.
  fn ← rp + zoneHeaderSize;
  zb[fn] ← ZoneInternal.NodeHeader[
    length: length - (zoneHeaderSize + usedNodeSize),
    extension: free[fwdp: freeList, backp: freeList]];
  -- set up allocated node (smallest possible) at end of block.
```



```
  an ← rp + (length - usedNodeSize);
  zb[an] ← ZoneInternal.NodeHeader[length: usedNodeSize, extension: inuse[]];
  -- set up the zone header
  z.rover ← fn;
  z.freeList ← freeList;
  z.length ← length;
  z.nextSegment ← nilSeg;
  z.zoneBase ← zb;
  z.threshold ← MAX[freeNodeSize, threshold];
  z.checking ← checking;
  z.node ← ZoneInternal.NodeHeader[
    length: 0, extension: free[fwdp: fn, backp: fn]];
  Process.InitializeMonitor[@z.LOCK];
  RETURN[zH: LOOPHOLE[z, Zone.Handle], s: okay]
  END;

-- returns the node pointed to by p to the free list

FreeNode: PUBLIC ENTRY PROC [zH: Zone.Handle, p: LONG POINTER]
  RETURNS [s: Zone.Status] =
  BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
  zb: ZoneInternal.Base = z.zoneBase;
  pp: LONG ORDERED POINTER = LOOPHOLE[p];
  node: ZoneInternal.NodePointer = Relative[zb, p - usedNodeSize];
  IF pp < zb OR pp > zb + LAST[CARDINAL] THEN RETURN[invalidNode];
  --Can't to a 16-bit relative pointer, => not from this zone.
  IF z.checking THEN
    BEGIN
    IF (s ← CheckZone[zH]) ~= okay THEN RETURN[invalidZone];
    IF (s ← CheckNode[zH, node]) ~= okay THEN RETURN[invalidNode];
    END;
  WITH zb[node] SELECT FROM
    free => RETURN[invalidNode];
    inuse =>
      BEGIN
      zb[node] ← ZoneInternal.NodeHeader[
        length, free[z.freeList, z.node.backp]];
      z.node.backp ← zb[z.node.backp].fwdp ← LOOPHOLE[node,
        ZoneInternal.FreeNodePointer];
      END;
    ENDCASE;
  RETURN[okay]
  END;

-- Returns the attributes of the zone (without checking)

GetAttributes: PUBLIC ENTRY PROC [zH: Zone.Handle]
  RETURNS [
    zoneBase: Environment.Base, threshold: Zone.BlockSize, checking: BOOLEAN,
    storage: LONG POINTER, length: Zone.BlockSize, next: Zone.SegmentHandle] =
  BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
  RETURN[
    zoneBase: z.zoneBase, threshold: z.threshold, checking: z.checking,
      storage: @z, length: z.length,
      next: LOOPHOLE[z.nextSegment, Zone.SegmentHandle]]
  END;

GetSegmentAttributes: PUBLIC ENTRY PROC [
```

```
    zH: Zone.Handle, sH: Zone.SegmentHandle]
    RETURNS [
      storage: LONG POINTER, length: Zone.BlockSize, next: Zone.SegmentHandle] =
    -- Returns the next segment in the chain, given sH; returns nullSegment if there is none. This p
**rocedure does not check the zone or segment for validity
    BEGIN
    OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer],
      seg: z.zoneBase[LOOPHOLE[sH, ZoneInternal.SegmentPointer]];
    sp: ZoneInternal.SegmentPointer = LOOPHOLE[sH];
    IF sH ~= nullSegment THEN
      RETURN[
        storage: @seg, length: seg.length,
          next: LOOPHOLE[seg.nextSegment, Zone.SegmentHandle]]
    END;

  -- Allocates a node of size n from the free list. The node is aligned on a multiple-word boundary
**determined by alignment. Search for a suitable node begins with z.rover; adjacent free nodes a
**re coalesced on the way; a large node is split in two if the remainder after allocating the node is
**larger than z.threshold

  MakeNode: PUBLIC ENTRY PROC [
    zH: Zone.Handle, n: Zone.BlockSize, alignment: Zone.Alignment]
    RETURNS [node: Environment.Base RELATIVE POINTER, s: Zone.Status] =
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    zb: ZoneInternal.Base = z.zoneBase;
    rover: ZoneInternal.FreeNodePointer ← z.rover;
    temp, neighbour: ZoneInternal.NodePointer;
    nodelength, nl: Zone.BlockSize;

    n ← MAX[n + usedNodeSize, freeNodeSize];
    --add the node overhead to the length of the request
    IF z.checking THEN IF (s ← CheckZone[zH]) ~= okay THEN RETURN[nil, s];
    --      start cycling through the free list
    DO
      nodelength ← zb[rover].length;
      --      first coalesce all free nodes adjacent to the rover
      FOR neighbour ← rover + nodelength, neighbour + nl DO
        WITH zb[neighbour] SELECT FROM
          inuse => EXIT;
          free =>
            BEGIN -- coalesce
            IF (nl ← length) = 0 THEN EXIT; -- end of zone
            zb[fwdp].backp ← backp;
            zb[backp].fwdp ← fwdp;
            z.rover ← rover; -- in case neighbor was z.rover
            nodelength ← nodelength + nl;
            END;
        ENDCASE;
      ENDLOOP;
    --      if the node pointed to by rover is big enough and alignment is not required, allocate it
    IF alignment = a1 AND nodelength >= n THEN
      BEGIN --        if it was too big, first split off the remainder
      IF (nl ← (nodelength - n)) > z.threshold THEN
        BEGIN -- split the block
        z.rover ← rover;
        zb[rover].length ← nl;
        temp ← LOOPHOLE[rover + nl, ZoneInternal.InuseNodePointer];
        nodelength ← n;
```



```
        END
      ELSE
        BEGIN
        zb[zb[rover].fwdp].backp ← zb[rover].backp;
        z.rover ← zb[zb[rover].backp].fwdp ← zb[rover].fwdp;
        temp ← LOOPHOLE[rover, ZoneInternal.InuseNodePointer];
        END;
      zb[temp] ← ZoneInternal.NodeHeader[nodelength, inuse[]];
      RETURN[
        LOOPHOLE[temp + usedNodeSize, Environment.Base RELATIVE POINTER], okay]
      END
      -- otherwise, either this node is not big enough or special alignment is required

    ELSE zb[rover].length ← nodelength;
    --      if multiple word alignment is required, see if we can carve up this node to fit
    IF alignment ~= a1 THEN
      BEGIN
      zr: LONG ORDERED POINTER = LOOPHOLE[@zb[rover]];
      mask: WORD = Inline.BITSHIFT[177777B, LOOPHOLE[alignment, INTEGER]];
      aPtr: MACHINE DEPENDENT RECORD [
        SELECT OVERLAID * FROM
          lp => [lp: LONG ORDERED POINTER],
          num => [lowbits, highbits: CARDINAL],
          ENDCASE];
      aPtr.lp ← zr + nodelength - n + usedNodeSize;
      --address of the node (without node overhead)
      aPtr.lowbits ← Inline.BITAND[aPtr.lowbits, mask];
      --address rounded down to be aligned
      aPtr.lp ← aPtr.lp - usedNodeSize; --subtract the node overhead again
      -- check if node fits exactly
      IF aPtr.lp = zr THEN
        BEGIN
        zb[zb[rover].fwdp].backp ← zb[rover].backp;
        z.rover ← zb[zb[rover].backp].fwdp ← zb[rover].fwdp;
        temp ← LOOPHOLE[rover, ZoneInternal.InuseNodePointer];
        END
        -- check if we can safely break off and free the front of the node

      ELSE
        IF aPtr.lp >= zr + z.threshold THEN
          BEGIN -- split the block
          diff: Inline.LongNumber;
          diff.lc ← aPtr.lp - zr;
          z.rover ← rover;
          zb[rover].length ← diff.lowbits;
          temp ← LOOPHOLE[rover + diff.lowbits, ZoneInternal.InuseNodePointer];
          nodelength ← nodelength - diff.lowbits;
          END
          -- otherwise, we failed to allocate from this node

        ELSE GO TO failure;
      zb[temp] ← ZoneInternal.NodeHeader[nodelength, inuse[]];
      [] ← Split[zH, temp, n];
      RETURN[
        LOOPHOLE[temp + usedNodeSize, Environment.Base RELATIVE POINTER], okay];
      EXITS failure => NULL
      END;
    -- Check to see if we have gone around the chain of free nodes
    IF (rover ← zb[rover].fwdp) = z.rover THEN EXIT;
    ENDLOOP;
```

```
    RETURN[nil, noRoomInZone];
    END;

  -- Returns the actual size of the allocated node (not including the overhead).  The node is not ch
**ecked for validity

  NodeSize: PUBLIC PROC [p: LONG POINTER] RETURNS [Zone.BlockSize] =
    BEGIN
    node: LONG POINTER TO ZoneInternal.NodeHeader = p - usedNodeSize;
    RETURN[node.length - usedNodeSize]
    END;

  LongPointerDifference: TYPE = MACHINE DEPENDENT RECORD [
    lowHalf: ZoneInternal.RPtr, highHalf: CARDINAL];

  -- computes a RELATIVE POINTER with respect to the zoneBase of zone z.  No check is made to
**see that p is within range of base
  Relative: PROC [base, p: LONG POINTER] RETURNS [ZoneInternal.RPtr] = INLINE {
    RETURN[LOOPHOLE[p - base, LongPointerDifference].lowHalf]};

  -- short description of what this procedure does

  RemoveSegment: PUBLIC ENTRY PROC [zH: Zone.Handle, sH: Zone.SegmentHandle]
    RETURNS [storage: LONG POINTER, s: Zone.Status] =
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    zb: ZoneInternal.Base = z.zoneBase;
    sp: ZoneInternal.SegmentPointer = LOOPHOLE[sH];
    lastNode: ZoneInternal.NodePointer =
      LOOPHOLE[sp + zb[sp].length - usedNodeSize];
    t: ZoneInternal.SegmentPointer;
    node: ZoneInternal.NodePointer;
    nl: Zone.BlockSize;
    storage ← @zb[sp];
    IF (s ← ValidateZone[zH]) ~= okay THEN RETURN[storage, s];
    IF z.checking THEN IF (s ← CheckZone[zH]) ~= okay THEN RETURN[storage, s];
    IF sp = nilSeg THEN RETURN[storage, invalidSegment];
    --      Check to see if segment is empty
    FOR node ← LOOPHOLE[sp + segmentHeaderSize, ZoneInternal.NodePointer],
      node + nl WHILE node < lastNode DO
      nl ← zb[node].length;
      WITH zb[node] SELECT FROM
        inuse => RETURN[storage, nonEmptySegment];
        ENDCASE;
      ENDLOOP; --      Remove segment from chain
    IF sp = z.nextSegment THEN z.nextSegment ← zb[sp].nextSegment
    ELSE
      FOR t ← z.nextSegment, zb[t].nextSegment WHILE t ~= nilSeg DO
        IF sp = zb[t].nextSegment THEN GOTO unlink;
        REPEAT
          unlink => zb[t].nextSegment ← zb[sp].nextSegment;
          --remove the segment from the chain

          FINISHED => RETURN[storage, invalidSegment];
        --OOPS! The segment was not part of this zone

        ENDLOOP; --      Remove all free nodes from free list
    FOR node ← LOOPHOLE[sp + segmentHeaderSize, ZoneInternal.NodePointer],
      node + nl WHILE node < lastNode DO
```



```
      nl ← zb[node].length;
      WITH zb[node] SELECT FROM
        inuse => EXIT;
        --as a result of previous checking, we are guaranteed to be at end of segment now

        free => BEGIN zb[backp].fwdp ← fwdp; zb[fwdp].backp ← backp; END;
        ENDCASE;
      ENDLOOP;
    z.rover ← z.node.fwdp; --reset rover to be outside segment
    RETURN[storage, okay];
    END;

  SetChecking: PUBLIC ENTRY PROC [zH: Zone.Handle, checking: BOOLEAN]
    RETURNS [s: Zone.Status] =
    -- sets the checking attribute of the zone;  if this is true, the zone is checked
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    z.checking ← checking;
    s ← IF checking THEN CheckZone[zH] ELSE okay;
    END;

  Split: PROC [zH: Zone.Handle, node: ZoneInternal.NodePointer, n: Zone.BlockSize]
    RETURNS [s: Zone.Status] = -- does the actual work of splitting a node
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    zb: ZoneInternal.Base = z.zoneBase;
    lastpart: ZoneInternal.FreeNodePointer;
    t: INTEGER;
    IF z.checking THEN IF (s ← CheckZone[zH]) ~= okay THEN RETURN[s];
    n ← MAX[n + usedNodeSize, freeNodeSize];
    --ensure that node remains large enough to free
    WITH zb[node] SELECT FROM
      free => RETURN[invalidNode];
      inuse =>
        IF (t ← zb[node].length - n) >= z.threshold THEN
          BEGIN --fabricate a free node out of the last part of the split node
          zb[node].length ← n; --adjust size of remaining node
          lastpart ← LOOPHOLE[node + n, ZoneInternal.FreeNodePointer];
          zb[lastpart] ← ZoneInternal.NodeHeader[
            length: t, extension: free[fwdp: z.freeList, backp: z.node.backp]];
          z.node.backp ← zb[zb[lastpart].backp].fwdp ← lastpart;
          END;
      ENDCASE;
    RETURN[okay]
    END;

  SplitNode: PUBLIC ENTRY PROC [
    zH: Zone.Handle, p: LONG POINTER, n: Zone.BlockSize]
    RETURNS [s: Zone.Status] =
    -- Splits a node, keeping the first n words and freeing the rest.
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    zb: ZoneInternal.Base = z.zoneBase;
    node: ZoneInternal.NodePointer = Relative[zb, p - usedNodeSize];
    RETURN[Split[zH, node, n]]
    END;

  ValidateZone: PROC [zH: Zone.Handle] RETURNS [Zone.Status] =
    -- Checks to see that the zone is well-formed
    BEGIN OPEN z: LOOPHOLE[zH, ZoneInternal.ZonePointer];
    zb: ZoneInternal.Base = z.zoneBase;
```

```
    lastNodePointer: LONG POINTER =
      LOOPHOLE[zH, ZoneInternal.ZonePointer] + z.length - usedNodeSize;
    -- to get around a compiler bug (this constant could be embedded in the declaration of the nex
**t one
    lastNode: ZoneInternal.NodePointer;
    sp: ZoneInternal.SegmentPointer; --  Check that primary storage of zone is okay
    lastNode ← Relative[zb, lastNodePointer];
    SELECT TRUE FROM
      (z.node.length ~= 0), (zb[lastNode].length ~= usedNodeSize),
        (zb[z.node.fwdp].backp ~= z.freeList OR zb[z.node.backp].fwdp ~=
          z.freeList) => RETURN[invalidZone];
      ENDCASE; --      Check that additional segments of zone are okay
    FOR sp ← z.nextSegment, zb[sp].nextSegment WHILE sp ~= nilSeg DO
      lastNode ← LOOPHOLE[sp + zb[sp].length - usedNodeSize,
        ZoneInternal.NodePointer];
      IF zb[lastNode].length ~= usedNodeSize THEN RETURN[invalidSegment]
      ENDLOOP;
    RETURN[okay]
    END;

  END.
LOG
Time: February 21, 1979  11:38 AM        By: Lauer          Action: Created file
Time: April 25, 1979  3:56 PM    By: Lauer          Action: Integrated into Pilot
Time: June 1, 1979  4:45 PM      By: Lauer          Action: Added facility for allocating aligned n
**odes (using the temporary interface ZoneExtensions)
Time: July 16, 1979  7:29 PM     By: Knutsen        Action: MakeNode now creates aligned nod
**es.  The temporary interface ZoneExtension is folded into ZoneImpl.
Time: April 7, 1980  2:30 PM     By: Luniewski      Action: Split off ZoneInternal.  Fixed ARs 285
**4 & 2892.
April 16, 1980  11:54 AM         By: Knutsen        Action: Now STARTed by InitializeZone.
September 19, 1980  5:03 PM      By: Forrest        Action: Convert to PlainText.
```

```
-- File: Boss.mesa,
--  Last Edit: BLyon  September 19, 1980  12:28 PM

DIRECTORY
  StatsDefs USING [StatIncr],
  CommUtilDefs USING [LockCode, UnlockCode, SetDebuggingPointer, CopyLong],
  DriverDefs USING [
    doDebug, doStats, Glitch, GiantVector, GetPupRouter, GetOisRouter,
    GetWordsPerIocb, SetWordsPerIocb, FreeQueueDestroy, FreeQueueMake,
    CreateDefaultEthernetOneDrivers, CreateDefaultEthernetDrivers, Network,
    DispatcherOff, DispatcherOn, DispatcherImpl, SystemBufferPoolImpl, QueueImpl],
  OISCPDefs USING [],
  PupDefs USING [],
  SpecialCommunication USING [PhysicalMedium],
  SpecialSystem USING [NetworkNumber];

Boss: MONITOR
  IMPORTS StatsDefs, CommUtilDefs, DriverDefs
  EXPORTS DriverDefs, OISCPDefs, PupDefs, SpecialCommunication
  SHARES DriverDefs =
  BEGIN OPEN CommUtilDefs, DriverDefs, SpecialSystem, SpecialCommunication;

  -- SemiPublic things for others
  firstNetwork: PUBLIC Network ← NIL;
  giantVector: PUBLIC POINTER TO GiantVector ← @bigBoy; -- Debugging only
  bigBoy: GiantVector;
  useCount: PUBLIC CARDINAL ← 0;

  state: {off, ready} ← off; -- on is ready with useCount>0

  IocbNotBigEnough: PUBLIC ERROR = CODE;
  CommPackageNotActive: PUBLIC ERROR = CODE;
  NetworkNonExistent: PUBLIC ERROR = CODE;

  GetUseCount: PUBLIC PROCEDURE RETURNS [CARDINAL] = BEGIN RETURN[useCount]; END;

  GetDoStats: PUBLIC PROCEDURE RETURNS [BOOLEAN] = BEGIN RETURN[doStats]; END;

  GetGiantVector: PUBLIC PROCEDURE RETURNS [POINTER TO GiantVector] =
    BEGIN RETURN[giantVector]; END;

  -- This code is a bit delicate.  There are probably many funny cases that won't work correctly.  In
**particular, there is a race condition between adding/deleting a driver and adding/deleting a ro
**uter.

  AddDeviceToChain: PUBLIC PROCEDURE [network: Network, iocbSize: CARDINAL] =
    BEGIN
    -- Add new drivers to the end of the chain so that the normal Ethernet driver will be network zer
**o.
    tail: Network ← firstNetwork;
    i: CARDINAL ← 1;
    UNTIL (iocbSize MOD 4) = 0 DO iocbSize ← iocbSize + 1; ENDLOOP;
    IF state = off THEN SetWordsPerIocb[MAX[GetWordsPerIocb[], iocbSize]]
    ELSE IF iocbSize > GetWordsPerIocb[] THEN Glitch[IocbNotBigEnough];
    IF state = ready THEN LockCode[network.interrupt];
    IF useCount > 0 THEN
      BEGIN
      network.activateDriver[];
```



```
      DriverDefs.GetPupRouter[].addNetwork[network];
      DriverDefs.GetOisRouter[].addNetwork[network];
      END;
    IF firstNetwork = NIL THEN
      BEGIN firstNetwork ← network; network.index ← 1; RETURN; END;
    UNTIL tail.next = NIL DO tail ← tail.next; i ← i + 1; ENDLOOP;
    tail.next ← network;
    network.index ← i + 1;
    END;

  RemoveDeviceFromChain: PUBLIC ENTRY PROCEDURE [network: Network] =
    BEGIN
    tail: Network ← firstNetwork;
    IF useCount > 0 THEN
      BEGIN
      DriverDefs.GetPupRouter[].removeNetwork[network];
      DriverDefs.GetOisRouter[].removeNetwork[network];
      network.deactivateDriver[];
      END;
    IF state = ready THEN UnlockCode[network.interrupt];
    IF firstNetwork = network THEN firstNetwork ← network.next
    ELSE
      BEGIN
      UNTIL tail.next = network DO tail ← tail.next; ENDLOOP;
      tail.next ← network.next;
      END;
    -- network.index is not updated.  It is used only to collect Gateway statistics.

    END;

  GetDeviceChain: PUBLIC ENTRY PROCEDURE RETURNS [Network] =
    BEGIN
    IF state # ready THEN Glitch[CommPackageNotActive];
    RETURN[firstNetwork];
    END;

  SmashDeviceChain: PUBLIC PROCEDURE = BEGIN firstNetwork ← NIL; END;

  -- The drivers may have to be told what its  network numbers are.
  -- This is certainly true if this is the first machine running on a network.
  -- Physical order is the location of the network on the network device chain .

  SetNetworkID: PUBLIC ENTRY PROCEDURE [
    physicalOrder: CARDINAL, medium: PhysicalMedium, newNetID: NetworkNumber]
    RETURNS [oldNetID: NetworkNumber] =
    BEGIN
    ENABLE UNWIND => NULL;
    net: Network;
    net ← GetNthDevice[physicalOrder, medium];
    oldNetID ← net.netNumber;
    net.netNumber ← newNetID;
    DriverDefs.GetPupRouter[].stateChanged[net];
    DriverDefs.GetOisRouter[].stateChanged[net];
    END;

  -- This procedure returns the OisNetID the Nth Network Object on the network device
  -- chain.  (where N = physicalOrder).

  GetNetworkID: PUBLIC ENTRY PROCEDURE [
```

```
  physicalOrder: CARDINAL, medium: PhysicalMedium] RETURNS [NetworkNumber] =
  BEGIN
  ENABLE UNWIND => NULL;
  RETURN[GetNthDevice[physicalOrder, medium].netNumber];
  END;

-- This procedure find the Nth (where N = physicalOrder) network of the specified medium
-- on the network device chain. This assumes that we locked (protected).

GetNthDevice: PRIVATE PROCEDURE [
  physicalOrder: CARDINAL, medium: PhysicalMedium] RETURNS [net: Network] =
  BEGIN
  i: CARDINAL ← 0;
  net ← firstNetwork;
  WHILE net # NIL DO
    IF net.device = medium THEN IF (i ← i + 1) = physicalOrder THEN RETURN;
    net ← net.next;
    ENDLOOP;
  ERROR NetworkNonExistent;
  END;

-- This may be called at any time. It does nothing if already ready.

OiscpPackageReady, PupPackageReady: PUBLIC ENTRY PROCEDURE =
  BEGIN CommPackageReady[]; END;

CommPackageReady: INTERNAL PROCEDURE =
  BEGIN
  network: Network;
  extra: CARDINAL ← 0;
  IF useCount = 0 AND firstNetwork = NIL THEN
    BEGIN
    [] ← CreateDefaultEthernetDrivers[];
    [] ← CreateDefaultEthernetOneDrivers[];
    END;
  IF state = ready THEN RETURN;
  -- On the Alto, MakeImage forgets low memory
  IF doDebug THEN CommUtilDefs.SetDebuggingPointer[giantVector];
  IF doStats THEN LockCode[StatsDefs.StatIncr];
  LockCode[DriverDefs.DispatcherImpl];
  LockCode[DriverDefs.SystemBufferPoolImpl];
  LockCode[DriverDefs.QueueImpl];
  LockCode[CommUtilDefs.CopyLong];
  FOR network ← firstNetwork, network.next UNTIL network = NIL DO
    extra ← extra + network.buffers; ENDLOOP;
  FreeQueueMake[extra];
  DispatcherOn[];
  FOR network ← firstNetwork, network.next UNTIL network = NIL DO
    LockCode[network.interrupt]; ENDLOOP;
  state ← ready;
  END;

CommPackageGo: PUBLIC ENTRY PROCEDURE =
  BEGIN
  network: Network;
  -- On the Alto, MakeImage forgets low memory
  IF doDebug THEN CommUtilDefs.SetDebuggingPointer[giantVector];
  CommPackageReady[];
  IF (useCount ← useCount + 1) > 1 THEN RETURN;
```



```
  FOR network ← firstNetwork, network.next UNTIL network = NIL DO
    network.activateDriver[]; ENDLOOP;
  END;

CommPackageOff: PUBLIC ENTRY PROCEDURE =
  BEGIN
  network: Network;
  IF (useCount ← useCount - 1) # 0 THEN RETURN;
  FOR network ← firstNetwork, network.next UNTIL network = NIL DO
    network.deactivateDriver[]; UnlockCode[network.interrupt]; ENDLOOP;
  DispatcherOff[];
  FreeQueueDestroy[];
  IF doStats THEN UnlockCode[StatsDefs.StatIncr];
  UnlockCode[CommUtilDefs.CopyLong];
  UnlockCode[DriverDefs.QueueImpl];
  UnlockCode[DriverDefs.SystemBufferPoolImpl];
  UnlockCode[DriverDefs.DispatcherImpl];
  state ← off;
  END;


-- initialization

IF doDebug THEN
  BEGIN
  CommUtilDefs.SetDebuggingPointer[giantVector];
  giantVector.slaThings ← giantVector.prThings ← NIL;
  END;
START DriverDefs.DispatcherImpl;
START DriverDefs.SystemBufferPoolImpl;
END.
```

LOG
August 1, 1980 11:23 AM, BLyon, Action: Added GetNetworkID, SetNetworkID, GetNthDevice.
September 8, 1980 2:53 AM, HGM, Action: Switch order of default Create for Ethernet1 + 2.
September 14, 1980 2:40 AM, HGM, Action: Count buffers needed for FreeQueueMake

```
-- BufferPoolImpl.mesa (last edited by: BLyon on: September 19, 1980 5:02 PM)

DIRECTORY
  BufferDefs USING [
    BufferPool, Buffer, BufferObject, Dequeue, Enqueue, OisBuffer, PupBuffer,
    RppBuffer, SppBuffer, QueueInitialize, QueueCleanup],
  CommUtilDefs USING [
    AllocateBuffers, FreeBuffers, LockBuffers, UnlockBuffers, Allocatelocbs,
    Freelocbs, GetReturnFrame, MaybeShorten],
  DriverTypes USING [bufferSeal, bufferPoolSeal, unsealed],
  DriverDefs USING [doDebug, doSee, doStats, Glitch, GetWordsPerlocb],
  Inline USING [BITAND, LowHalf],
  OISCPDefs, -- EXPORTS
  PupDefs, -- EXPORTS
  Process USING [InitializeMonitor, InitializeCondition, MsecToTicks],
  StatsDefs USING [StatIncr];

BufferPoolImpl: MONITOR LOCKS pool USING pool: BufferDefs.BufferPool
  IMPORTS Inline, Process, StatsDefs, BufferDefs, CommUtilDefs, DriverDefs
  EXPORTS BufferDefs, DriverDefs, OISCPDefs
  SHARES BufferDefs =
  BEGIN OPEN BufferDefs, DriverDefs;

  -- size of a buffer without any data
  rawOverhead: CARDINAL = SIZE[raw BufferDefs.BufferObject];
  -- 1 extra for pup checksum
  pupOverhead: CARDINAL = SIZE[pupWords pup BufferDefs.BufferObject] + 1;
  overhead: CARDINAL = MAX[
    pupOverhead, SIZE[rppWords rpp BufferDefs.BufferObject] + 1, SIZE[oisWords ois
      BufferDefs.BufferObject], SIZE[sppWords spp ois BufferDefs.BufferObject]];

  -- for the Glitches
  BufferPoolSealBroken: PUBLIC ERROR = CODE;
  PoolSealBroken: PUBLIC ERROR = CODE;
  BufferPoolNotInitialized: PUBLIC ERROR = CODE;
  QueueScrambled: PUBLIC ERROR = CODE;

  -- Cool procedures

  -- The caller must have the total, reserve, send, receive, dataWordsPerBuffer, and
  -- the bufferSize fields filled in.  All other fields will be filled in by this routine.
  BufferPoolMake: PUBLIC PROCEDURE [pool: BufferPool] =
    BEGIN
    b: Buffer;
    i: CARDINAL;
    pool.seal ← DriverTypes.bufferPoolSeal;
    pool.firstBuffer ← NIL;
    pool.receiveInUse ← pool.sendInUse ← 0;
    -- 2 for checksum, 4 for end test, 3 for round down
    pool.wordsPerBuffer ← pool.dataWordsPerBuffer + overhead;
    IF (pool.bufferSize = big) THEN
      -- 2 for checksum, 4 for end test, 3 for round down
      pool.wordsPerBuffer ← pool.wordsPerBuffer + 2 + 4 + 3;
    UNTIL Inline.BITAND[pool.wordsPerBuffer, 3] = 0 DO
      pool.wordsPerBuffer ← pool.wordsPerBuffer + 1; ENDLOOP;
    -- all buffers are quad-word aleigned
    Process.InitializeMonitor[CommUtilDefs.MaybeShorten[@pool.LOCK]];
    Process.InitializeCondition[
```



```
      CommUtilDefs.MaybeShorten[@pool.freeQueueNotEmpty], Process.MsecToTicks[
      10000]];
    Process.InitializeCondition[
      CommUtilDefs.MaybeShorten[@pool.sendBufferAvailable], Process.MsecToTicks[
      10000]];
    Process.InitializeCondition[
      CommUtilDefs.MaybeShorten[@pool.receiveBufferAvailable],
      Process.MsecToTicks[10000]];
    pool.firstBuffer ← CommUtilDefs.AllocateBuffers[
      pool.wordsPerBuffer*pool.total + 3];
    CommUtilDefs.LockBuffers[pool.firstBuffer];
    IF (pool.bufferSize = big) THEN
      BEGIN
      -- This determines the buffer alignment: see DriverTypes.Encapsulation
      UNTIL Inline.BITAND[Inline.LowHalf[@pool.firstBuffer.encapsulation], 3] = 3
        DO pool.firstBuffer ← pool.firstBuffer + 1; ENDLOOP;
      END;
    QueueInitialize[CommUtilDefs.MaybeShorten[@pool.freeQueue]];
    b ← pool.firstBuffer;
    FOR i IN [0..pool.total) DO
      Process.InitializeCondition[
        CommUtilDefs.MaybeShorten[@b.sendCompleted], Process.MsecToTicks[10000]];
      b.iocbChain ← NIL;
      b.userData ← NIL;
      b.userPtr ← NIL;
      b.allNets ← b.bypassZeroNet ← FALSE;
      b.type ← raw;
      b.size ← pool.bufferSize;
      b.pupLength ← b.length ← 0;
      b.bufferNumber ← i;
      b.pupType ← last;
      b.queue ← NIL;
      b.pool ← pool;
      b.next ← NIL;
      IF doDebug THEN b.seal ← DriverTypes.bufferSeal;
      b.requeueProcedure ← ReturnBufferToCorrectPool;
      Enqueue[@pool.freeQueue, b];
      b ← b + pool.wordsPerBuffer;
      ENDLOOP;
    IF (pool.bufferSize = big) AND (GetWordsPerlocb[] # 0) THEN
      BEGIN
      iocb: LONG POINTER;
      wordsPerlocb: CARDINAL ← GetWordsPerlocb[];
      iocb ← CommUtilDefs.Allocatelocbs[wordsPerlocb*pool.total + 3];
      b ← pool.firstBuffer;
      FOR i IN [0..pool.total) DO
        -- this does NOT align each individual iocb
        b.iocbChain ← iocb;
        iocb ← iocb + wordsPerlocb;
        b ← b + pool.wordsPerBuffer;
        ENDLOOP;
      END;
    END; -- BufferPoolMake

  BufferPoolDestroy: PUBLIC PROCEDURE [pool: BufferPool] =
    BEGIN
    UNTIL Dequeue[@pool.freeQueue] = NIL DO ENDLOOP;
    QueueCleanup[@pool.freeQueue];
```

```
  IF pool.bufferSize = big THEN CommUtilDefs.UnlockBuffers[pool.firstBuffer];
  IF pool.firstBuffer.iocbChain # NIL THEN
    CommUtilDefs.FreeIocbs[pool.firstBuffer.iocbChain];
  CommUtilDefs.FreeBuffers[pool.firstBuffer];
  pool.firstBuffer ← NIL;
  IF doDebug THEN pool.seal ← DriverTypes.unsealed;
  END; -- BufferPoolDestroy

BuffersLeftInPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool]
  RETURNS [CARDINAL] =
  BEGIN
  IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
  RETURN[pool.freeQueue.length];
  END; -- BufferLeftInPool

SendBuffersLeftInPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool]
  RETURNS [CARDINAL] =
  BEGIN
  IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
  RETURN[pool.send - pool.sendInUse];
  END; -- SendBufferLeftInPool

ReceiveBuffersLeftInPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool]
  RETURNS [CARDINAL] =
  BEGIN
  IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
  RETURN[pool.receive - pool.receiveInUse];
  END; -- ReceiveBufferLeftInPool

EnumerateBuffersInPool: PUBLIC PROCEDURE [
  pool: BufferPool, proc: PROCEDURE [Buffer]] =
  BEGIN
  b: BufferDefs.Buffer ← pool.firstBuffer;
  i: CARDINAL;
  IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[BufferPoolSealBroken];
  FOR i IN [0..pool.total) DO proc[b]; b ← b + pool.wordsPerBuffer; ENDLOOP;
  END; -- EnumerateBuffersInPool

ReturnPupBufferToPool: PUBLIC PROCEDURE [BufferPool, PupBuffer] =
  LOOPHOLE[ReturnBufferToPool];
ReturnRppBufferToPool: PUBLIC PROCEDURE [BufferPool, RppBuffer] =
  LOOPHOLE[ReturnBufferToPool];
ReturnOisBufferToPool: PUBLIC PROCEDURE [BufferPool, OisBuffer] =
  LOOPHOLE[ReturnBufferToPool];
ReturnSppBufferToPool: PUBLIC PROCEDURE [BufferPool, SppBuffer] =
  LOOPHOLE[ReturnBufferToPool];
ReturnBufferToPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool, b: Buffer] =
  BEGIN
  IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
  IF doDebug AND b.pool # pool THEN Glitch[QueueScrambled];
  -- Note: we do some "initialization" of things here since there are several ways to get
  -- buffers from the freeQueue.
```



```
  b.requeueProcedure ← ReturnBufferToCorrectPool;
  b.network ← NIL;
  b.userPtr ← b.userData ← NIL;
  Enqueue[@pool.freeQueue, b];
  -- This is ugly, butthere isn't any way to make things work without it if 2 PROCESSes
  -- are waiting for clumps.
  BROADCAST pool.freeQueueNotEmpty;
  END; -- ReturnBufferToPool

-- Hot procedures

GetFreeSendBufferFromPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool]
  RETURNS [b: Buffer] =
  BEGIN
  IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
  IF doStats AND ~pool.sendInUse < pool.send THEN
    StatsDefs.StatIncr[statBufferWaits];
  UNTIL (pool.sendInUse < pool.send) DO WAIT pool.sendBufferAvailable; ENDLOOP;
  b ← Dequeue[@pool.freeQueue];
  IF doDebug AND b = NIL THEN Glitch[QueueScrambled];
  IF doDebug AND b.pool # pool THEN Glitch[QueueScrambled];
  IF doDebug OR doSee THEN b.type ← raw;
  IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
  b.requeueProcedure ← ReturnSendBufferToCorrectPool;
  pool.sendInUse ← pool.sendInUse + 1;
  END; -- GetFreeSendBufferFromPool

GetFreeReceiveBufferFromPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool]
  RETURNS [b: Buffer] =
  BEGIN
  IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
  IF doStats AND ~pool.receiveInUse < pool.receive THEN
    StatsDefs.StatIncr[statBufferWaits];
  UNTIL (pool.receiveInUse < pool.receive) DO
    WAIT pool.receiveBufferAvailable; ENDLOOP;
  b ← Dequeue[@pool.freeQueue];
  IF doDebug AND b = NIL THEN Glitch[QueueScrambled];
  IF doDebug AND b.pool # pool THEN Glitch[QueueScrambled];
  IF doDebug OR doSee THEN b.type ← raw;
  IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
  b.requeueProcedure ← ReturnReceiveBufferToCorrectPool;
  pool.receiveInUse ← pool.receiveInUse + 1;
  END; -- GetFreeReceiveBufferFromPool

-- Get a free buffer, but don't wait if there is none

MaybeGetFreeSendBufferFromPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool]
  RETURNS [b: Buffer] =
  BEGIN
  IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
  IF doStats AND ~pool.sendInUse < pool.send THEN
    StatsDefs.StatIncr[statBufferWaits];
```

```
    IF (pool.sendInUse < pool.send) THEN
      BEGIN
      b ← Dequeue[@pool.freeQueue];
      IF doDebug AND b = NIL THEN Glitch[QueueScrambled];
      IF doDebug AND b.pool # pool THEN Glitch[QueueScrambled];
      IF doDebug OR doSee THEN b.type ← raw;
      IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
      b.requeueProcedure ← ReturnSendBufferToCorrectPool;
      pool.sendInUse ← pool.sendInUse + 1;
      END
    ELSE b ← NIL;
    END; -- MaybeGetFreeSendBufferFromPool

  MaybeGetFreeReceiveBufferFromPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool]
    RETURNS [b: Buffer] =
    BEGIN
    IF doDebug AND pool.seal # DriverTypes.bufferPoolSeal THEN
      Glitch[PoolSealBroken];
    IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
    IF doStats AND ~pool.receiveInUse < pool.receive THEN
      StatsDefs.StatIncr[statBufferWaits];
    IF (pool.receiveInUse < pool.receive) THEN
      BEGIN
      b ← Dequeue[@pool.freeQueue];
      IF doDebug AND b = NIL THEN Glitch[QueueScrambled];
      IF doDebug AND b.pool # pool THEN Glitch[QueueScrambled];
      IF doDebug OR doSee THEN b.type ← raw;
      IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
      b.requeueProcedure ← ReturnReceiveBufferToCorrectPool;
      pool.receiveInUse ← pool.receiveInUse + 1;
      END
    ELSE b ← NIL;
    END; -- MaybeGetFreeReceiveBufferFromPool

  GetFreeSendOisBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: OisBuffer] =
    BEGIN
    b ← LOOPHOLE[GetFreeSendBufferFromPool[pool], OisBuffer];
    IF doDebug OR doSee THEN b.type ← ois;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
    END; -- GetFreeSendOisBufferFromPool

  GetFreeReceiveOisBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: OisBuffer] =
    BEGIN
    b ← LOOPHOLE[GetFreeReceiveBufferFromPool[pool], OisBuffer];
    IF doDebug OR doSee THEN b.type ← ois;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
    END; -- GetFreeReceiveOisBufferFromPool

  GetFreeSendSppBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: SppBuffer] =
    BEGIN
    b ← LOOPHOLE[GetFreeSendBufferFromPool[pool], SppBuffer];
    IF doDebug OR doSee THEN b.type ← ois;
    b.systemPacket ← b.sendAck ← b.attention ← FALSE;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
    END; -- GetFreeSendSppBufferFromPool
```



```
  GetFreeReceiveSppBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: SppBuffer] =
    BEGIN
    b ← LOOPHOLE[GetFreeReceiveBufferFromPool[pool], SppBuffer];
    IF doDebug OR doSee THEN b.type ← ois;
    b.systemPacket ← b.sendAck ← b.attention ← FALSE;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
    END; -- GetFreeReceiveSppBufferFromPool

  MaybeGetFreeSendOisBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: OisBuffer] =
    BEGIN
    b ← LOOPHOLE[MaybeGetFreeSendBufferFromPool[pool], OisBuffer];
    IF b # NIL THEN BEGIN IF doDebug OR doSee THEN b.type ← ois; END;
    END; -- MaybeGetFreeSendOisBufferFromPool

  MaybeGetFreeReceiveOisBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: OisBuffer] =
    BEGIN
    b ← LOOPHOLE[MaybeGetFreeReceiveBufferFromPool[pool], OisBuffer];
    IF b # NIL THEN BEGIN IF doDebug OR doSee THEN b.type ← ois; END;
    END; -- MaybeGetFreeReceiveOisBufferFromPool

  MaybeGetFreeSendSppBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: SppBuffer] =
    BEGIN
    b ← LOOPHOLE[MaybeGetFreeSendBufferFromPool[pool], SppBuffer];
    IF b # NIL THEN
      BEGIN
      IF doDebug OR doSee THEN b.type ← ois;
      b.systemPacket ← b.sendAck ← b.attention ← FALSE;
      END;
    END; -- MaybeGetFreeSendSppBufferFromPool

  MaybeGetFreeReceiveSppBufferFromPool: PUBLIC PROCEDURE [pool: BufferPool]
    RETURNS [b: SppBuffer] =
    BEGIN
    b ← LOOPHOLE[MaybeGetFreeReceiveBufferFromPool[pool], SppBuffer];
    IF b # NIL THEN
      BEGIN
      IF doDebug OR doSee THEN b.type ← ois;
      b.systemPacket ← b.sendAck ← b.attention ← FALSE;
      END;
    END; -- MaybeGetFreeReceiveSppBufferFromPool

  ReturnSendPupBufferToPool: PUBLIC PROCEDURE [BufferPool, PupBuffer] =
    LOOPHOLE[ReturnSendBufferToPool];
  ReturnSendRppBufferToPool: PUBLIC PROCEDURE [BufferPool, RppBuffer] =
    LOOPHOLE[ReturnSendBufferToPool];
  ReturnSendOisBufferToPool: PUBLIC PROCEDURE [BufferPool, OisBuffer] =
    LOOPHOLE[ReturnSendBufferToPool];
  ReturnSendSppBufferToPool: PUBLIC PROCEDURE [BufferPool, SppBuffer] =
    LOOPHOLE[ReturnSendBufferToPool];
  ReturnSendBufferToPool: PUBLIC ENTRY PROCEDURE [pool: BufferPool, b: Buffer] =
    BEGIN
    IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
    IF doDebug AND b.pool # pool THEN Glitch[QueueScrambled];
    -- Note: we do some "initialization" of things here since there are several ways to get
    -- buffers from the freeQueue.
```

```
    b.network ← NIL;
    b.userPtr ← b.userData ← NIL;
    Enqueue[@pool.freeQueue, b];
    pool.sendInUse ← pool.sendInUse - 1;
    -- This is ugly, butthere isn't any way to make things work without it if 2 PROCESSes
    -- are waiting for clumps.
    BROADCAST pool.sendBufferAvailable;
    END; -- ReturnSendBufferToPool

  ReturnReceivePupBufferToPool: PUBLIC PROCEDURE [BufferPool, PupBuffer] =
    LOOPHOLE[ReturnReceiveBufferToPool];
  ReturnReceiveRppBufferToPool: PUBLIC PROCEDURE [BufferPool, RppBuffer] =
    LOOPHOLE[ReturnReceiveBufferToPool];
  ReturnReceiveOisBufferToPool: PUBLIC PROCEDURE [BufferPool, OisBuffer] =
    LOOPHOLE[ReturnReceiveBufferToPool];
  ReturnReceiveSppBufferToPool: PUBLIC PROCEDURE [BufferPool, SppBuffer] =
    LOOPHOLE[ReturnReceiveBufferToPool];
  ReturnReceiveBufferToPool: PUBLIC ENTRY PROCEDURE [
    pool: BufferPool, b: Buffer] =
    BEGIN
    IF doDebug AND pool.firstBuffer = NIL THEN Glitch[BufferPoolNotInitialized];
    IF doDebug AND b.pool # pool THEN Glitch[QueueScrambled];
    -- Note: we do some "initialization" of things here since there are several ways to get
    -- buffers from the freeQueue.
    b.network ← NIL;
    b.userPtr ← b.userData ← NIL;
    Enqueue[@pool.freeQueue, b];
    pool.receiveInUse ← pool.receiveInUse - 1;
    -- This is ugly, butthere isn't any way to make things work without it if 2 PROCESSes
    -- are waiting for clumps.
    BROADCAST pool.receiveBufferAvailable;
    END; -- ReturnReceiveBufferToPool

  ReturnBufferToCorrectPool: PUBLIC PROCEDURE [b: Buffer] =
    BEGIN ReturnBufferToPool[b.pool, b]; END; -- ReturnBufferToCorrectPool

  ReturnSendBufferToCorrectPool: PUBLIC PROCEDURE [b: Buffer] =
    BEGIN ReturnSendBufferToPool[b.pool, b]; END; -- ReturnSendBufferToCorrectPool

  ReturnReceiveBufferToCorrectPool: PUBLIC PROCEDURE [b: Buffer] =
    BEGIN ReturnReceiveBufferToPool[b.pool, b]; END;
  -- ReturnReceiveBufferToCorrectPool

  -- initialization

  END. -- BufferPoolImpl module
```

LOG

Time: April 17, 1980 2:54 PM By: Dalal Action: created file for Pilot 5.0.
Time: April 17, 1980 2:54 PM By: Dalal Action: merged PoolCool and PoolHot.
Time: September 17, 1980 4:17 PM By: BLyon Action: added 2 + 4 + 3 to size of big buffers.

```
-- ChecksumsImpl.mesa (last edit by: BLyon/HGM August 20, 1980 10:57 AM)

DIRECTORY
  BufferDefs USING [OisBuffer, OisBufferObject],
  Checksums USING [ComputeChecksum],
  CommunicationInternal USING [],
  CommUtilDefs USING [thisIsAnAlto],
  Inline USING [BITAND, BITNOT, BITOR, BITSHIFT],
  Mopcodes USING [zADD, zDADD, zDUP, zEXCH, zJNE3, zLIN1, zLI0, zNOOP, zSUB],
  OISCPTypes USING [bytesPerOisPktHeader];

ChecksumsImpl: PROGRAM
  IMPORTS Checksums, Inline
  EXPORTS CommunicationInternal, Checksums
  SHARES BufferDefs =

  BEGIN OPEN Checksums;

  -- These procedures sets the checksum field of the Ois Packet.
  SetChecksum: PUBLIC PROCEDURE [b: BufferDefs.OisBuffer] =
    BEGIN
    cs: CARDINAL ← 0;
    IF b.size = user AND b.userData # NIL THEN
      BEGIN
      cs ←
        IF CommUtilDefs.thisIsAnAlto THEN SlowlyComputeChecksum[
        cs, OISCPTypes.bytesPerOisPktHeader/2 - 1, @b.oisPktLength]
        ELSE ComputeChecksum[
          cs, OISCPTypes.bytesPerOisPktHeader/2 - 1, @b.oisPktLength];
      cs ←
        IF CommUtilDefs.thisIsAnAlto THEN SlowlyComputeChecksum[
        cs, (b.userDataLength + 1)/2, b.userData]
        ELSE ComputeChecksum[cs, (b.userDataLength + 1)/2, b.userData];
      END
    ELSE
      cs ←
        IF CommUtilDefs.thisIsAnAlto THEN SlowlyComputeChecksum[
        cs, (b.oisPktLength + 1)/2 - 1, @b.oisPktLength]
        ELSE ComputeChecksum[cs, (b.oisPktLength + 1)/2 - 1, @b.oisPktLength];
    b.checksum ← cs;
    END;

  -- These procedures checks the checksum field of the Ois Packet,
  -- and returns TRUE or FALSE. The buffer is always a system buffer

  TestChecksum: PUBLIC PROCEDURE [b: BufferDefs.OisBuffer] RETURNS [BOOLEAN] =
    BEGIN
    cs: CARDINAL ← 0;
    IF b.checksum = 177777B THEN RETURN[TRUE];
    cs ←
      IF CommUtilDefs.thisIsAnAlto THEN SlowlyComputeChecksum[
      cs, (b.oisPktLength + 1)/2 - 1, @b.oisPktLength]
      ELSE ComputeChecksum[cs, (b.oisPktLength + 1)/2 - 1, @b.oisPktLength];
    RETURN[b.checksum = cs];
    END;

  SlowlyComputeChecksum: PUBLIC PROCEDURE [
    cs: CARDINAL, nWords: CARDINAL, p: LONG POINTER] RETURNS [CARDINAL] =
    BEGIN
```



```
    t: CARDINAL;
    THROUGH [0..nWords) DO
      IF cs > (t ← cs + p↑) THEN cs ← t + 1 ELSE cs ← t;
      IF cs >= 100000B THEN cs ← cs*2 + 1 ELSE cs ← cs*2;
      p ← p + 1;
      ENDLOOP;
    IF cs = 177777B THEN cs ← 0;
    RETURN[cs];
    END;

  -- This procedure updates the checksum field of the Ois Packet
  -- after it increments the oisTransportControl.
  -- Assumptions made about an OisBuffer:
  -- checksums is the first word of the ois varient.
  -- oisPktLength is the length of the ois varient (in Bytes).
  -- The work containing transportControl is located 1 after oisPktLength.

  dummyBuf: PRIVATE POINTER TO BufferDefs.OisBufferObject = LOOPHOLE[0];
  relativeChecksumLoc: PRIVATE POINTER = @dummyBuf.checksum;
  relativeOisTransportControlLoc: PRIVATE POINTER = @dummyBuf.oisPktLength + 1;
  transportControlOffset: INTEGER =
    relativeOisTransportControlLoc - relativeChecksumLoc;
  IncrOisTransportControlAndUpdateChecksum: PUBLIC PROCEDURE [
    b: BufferDefs.OisBuffer] =
    BEGIN
    newValLoc: LONG POINTER TO WORD =
      LOOPHOLE[b + LOOPHOLE[relativeOisTransportControlLoc, CARDINAL]];
    oldVal: WORD ← newValLoc↑;
    resisdual: INTEGER;
    b.oisTransportControl ← b.oisTransportControl + 1;
    IF b.checksum = 177777B THEN RETURN;
    resisdual ← Inline.BITAND[
      (b.oisPktLength + 1)/2 - transportControlOffset, 17B];
    b.checksum ← OnesAdd[
      b.checksum, LeftCycle[OnesSub[newValLoc↑, oldVal], resisdual]];
    END; -- UpdateChecksum

  LeftCycle: PROCEDURE [val: CARDINAL, dist: INTEGER] RETURNS [CARDINAL] = INLINE
    BEGIN OPEN Inline;
    RETURN[BITOR[BITSHIFT[val, dist], BITSHIFT[val, dist - 16]]];
    END;

  OnesSub: PROCEDURE [a, b: CARDINAL] RETURNS [CARDINAL] = INLINE
    BEGIN RETURN[OnesAdd[a, Inline.BITNOT[b]]]; END;

  OnesAdd: PROCEDURE [a, b: CARDINAL] RETURNS [c: CARDINAL] = MACHINE CODE
    BEGIN
    -- c ← a + b; IF c<a THEN c ← c + 1; IF c = 177777B THEN c ← 0;
    Mopcodes.zLI0;
    Mopcodes.zEXCH;
    Mopcodes.zLI0;
    Mopcodes.zDADD;
    Mopcodes.zADD;
    Mopcodes.zDUP;
    Mopcodes.zLIN1;
    Mopcodes.zJNE3;
    Mopcodes.zDUP;
    Mopcodes.zSUB;
```

```
    Mopcodes.zNOOP; -- not clear if this is really needed.
    END;
END. -- ChecksumsImpl
```

```
-- CommunicationControl.mesa (last edited by: HGM on: September 7, 1980 12:49 PM)
-- Function: The control module for Pilot's Communication configuration.

DIRECTORY
  CommunicationInternal USING [
    ChecksumsImpl, EchoServerImpl, PilotCommUtil, RouterImpl, RoutingTableImpl,
    SocketImpl, NetworkStreamMgr, PacketStreamMgr],
  CommunicationPrograms USING [],
  DriverDefs USING [CommPackageGo, CommPackageOff, Boss],
  Echo USING [CreateServer, DeleteServer],
  OISCPDefs USING [OiscpPackageReady],
  Router USING [OisRouterOff, OisRouterOn],
  SocketInternal USING [SocketOn],
  StatsOps USING [StatsHot];

CommunicationControl: MONITOR
  IMPORTS
    CommunicationInternal, DriverDefs, Echo, OISCPDefs, Router, SocketInternal,
    StatsOps
  EXPORTS OISCPDefs, CommunicationPrograms =
  BEGIN

  -- monitor data
  useCount: CARDINAL;

  StartupCommunication: PROCEDURE =
    -- module initialization:
    BEGIN
    useCount ← 0;
    START StatsOps.StatsHot;
    START CommunicationInternal.PilotCommUtil;
    -- AdjustSystemBufferParms[];
    START DriverDefs.Boss;
    START CommunicationInternal.ChecksumsImpl;
    START CommunicationInternal.RouterImpl;
    START CommunicationInternal.RoutingTableImpl;
    START CommunicationInternal.SocketImpl;
    START CommunicationInternal.EchoServerImpl;
    OiscpPackageMake[];
    START CommunicationInternal.PacketStreamMgr;
    START CommunicationInternal.NetworkStreamMgr;
    END;

  InitializeCommunication: PUBLIC PROCEDURE =
    -- module initialization:
    BEGIN
    -- the start trap takes care of everything

    END;


  -- The procedure that a normal client calls to start things either before or
  -- after the MakeImage.

  OiscpPackageMake: PUBLIC ENTRY PROCEDURE =
    BEGIN
    IF (useCount ← useCount + 1) > 1 THEN RETURN;
    OISCPDefs.OiscpPackageReady[];
    DriverDefs.CommPackageGo[];
```



```
    Router.OisRouterOn[];
    SocketInternal.SocketOn[];
    Echo.CreateServer[];
    END;

  -- The procedure that a normal client calls to stop things

  OiscpPackageDestroy: PUBLIC ENTRY PROCEDURE =
    BEGIN
    IF (useCount ← useCount - 1) > 0 THEN RETURN;
    Echo.DeleteServer[];
    Router.OisRouterOff[];
    DriverDefs.CommPackageOff[];
    END;

  -- main line code

  StartupCommunication[];

  END. -- CommunicationControl module
LOG
Time: January 20, 1980 11:01 AM          By: Dalal  Action: changed initialization order.
Time: February 5, 1980 8:31 PM           By: Dalal  Action: started of SimpleHeapImpl.
Time: February 5, 1980 9:16 PM           By: Dalal  Action: added Echo Server.
Time: August 18, 1978 11:37 AM           By: Knutsen          Action: Module now STARTed by
**InitializeCommunication[].
Time: May 6, 1980 6:35 PM      By: BLyon Action: Temp removed CommunicationPrograms from
**EXPORTS; added mainline code so that Communications can be build/started seperately from
**TestPilotKernel .. tajo.
Time: August 11, 1980 3:39 PM By: BLyon Action: made it so that communication can start on a st
**art trap (added running: BOOLEAN).
Time: August 20, 1980 10:38 AM           By: BLyon Action: added START ChecksumsImpl
```

```
-- DispatcherImpl.mesa (last edited by: HGM on: September 13, 1980 3:18 PM)

DIRECTORY
  BufferDefs USING [
    Buffer, Dequeue, Enqueue, QueueCleanup, QueueInitialize, QueueLength,
    QueueObject],
  DriverDefs USING [
    doStats, doDebug, GetGiantVector, Glitch, Network, ReturnFreeBuffer, Router,
    RouterObject],
  Process USING [SetTimeout, MsecToTicks],
  StatsDefs USING [StatIncr];

DispatcherImpl: MONITOR
  IMPORTS BufferDefs, DriverDefs, Process, StatsDefs
  EXPORTS DriverDefs
  SHARES BufferDefs =
  BEGIN

  mainFork: PROCESS;
  dispatcherPleaseDie: BOOLEAN;
  dispatcherReady: CONDITION;
  globalInputQueue: BufferDefs.QueueObject;
  globalOutputQueue: BufferDefs.QueueObject;
  pupRouter, oisRouter: PUBLIC DriverDefs.Router;
  dummyRouter: DriverDefs.Router;

  dummyRouterObject: DriverDefs.RouterObject ←
    [input: DummyInputer, broadcast: DummyBroadcaster, addNetwork: DummyAddDelete,
      removeNetwork: DummyAddDelete, stateChanged: DummyStateChanged];

  UnknownDecapsulation: ERROR = CODE;

  -- Cold procedures

  DummyInputer: PROCEDURE [b: BufferDefs.Buffer] =
    BEGIN
    IF DriverDefs.doStats THEN StatsDefs.StatIncr[statPacketsDiscarded];
    b.requeueProcedure[b];
    END;

  DummyBroadcaster: PROCEDURE [b: BufferDefs.Buffer] =
    BEGIN b.requeueProcedure[b]; END;

  DummyAddDelete: PROCEDURE [DriverDefs.Network] = BEGIN END;

  DummyStateChanged: PROCEDURE [DriverDefs.Network] = BEGIN END;

  SetPupRouter: PUBLIC PROCEDURE [router: DriverDefs.Router] =
    BEGIN
    IF router = NIL THEN router ← @dummyRouterObject;
    pupRouter ← router;
    END;

  SetOisRouter: PUBLIC PROCEDURE [router: DriverDefs.Router] =
    BEGIN
    IF router = NIL THEN router ← @dummyRouterObject;
    oisRouter ← router;
    END;
```



```
  GetPupRouter: PUBLIC PROCEDURE RETURNS [DriverDefs.Router] =
    BEGIN RETURN[pupRouter]; END;

  GetOisRouter: PUBLIC PROCEDURE RETURNS [DriverDefs.Router] =
    BEGIN RETURN[oisRouter]; END;

  DispatcherOn: PUBLIC PROCEDURE =
    BEGIN
    dispatcherPleaseDie ← FALSE;
    BufferDefs.QueueInitialize[@globalInputQueue];
    BufferDefs.QueueInitialize[@globalOutputQueue];
    mainFork ← FORK MainDispatcher[];
    END;

  DispatcherOff: PUBLIC PROCEDURE =
    BEGIN
    DispatcherOffLocked[];
    JOIN mainFork;
    BufferDefs.QueueCleanup[@globalInputQueue];
    BufferDefs.QueueCleanup[@globalOutputQueue];
    END;

  DispatcherOffLocked: ENTRY PROCEDURE = INLINE
    BEGIN dispatcherPleaseDie ← TRUE; NOTIFY dispatcherReady; END;

  -- Hot procedures

  MainDispatcher: PUBLIC PROCEDURE =
    BEGIN
    b: BufferDefs.Buffer;
    network: DriverDefs.Network;
    UNTIL dispatcherPleaseDie DO
      WHILE BufferDefs.QueueLength[@globalOutputQueue] # 0 DO
        -- give back free buffers first
        b ← GrabOutputBuffer[];
        IF b.allNets THEN SendToNextNetwork[b] ELSE b.requeueProcedure[b];
        ENDLOOP;
      IF BufferDefs.QueueLength[@globalInputQueue] # 0 THEN
        BEGIN
        b ← GrabInputBuffer[];
        network ← b.network;
        -- give it to the right router, and it requeues the buffer
        SELECT network.decapsulateBuffer[b] FROM
          pup, rpp =>
            BEGIN
            IF DriverDefs.doDebug THEN b.type ← pup;
            pupRouter.input[b];
            END;
          ois =>
            BEGIN
            IF DriverDefs.doDebug THEN b.type ← ois;
            oisRouter.input[b];
            END;
          rejected => dummyRouter.input[b];
          processed => NULL;
          ENDCASE => DriverDefs.Glitch[UnknownDecapsulation];
        LOOP;
        END;
      Wait[];
```

```
    ENDLOOP;
  END;

GrabOutputBuffer: ENTRY PROCEDURE RETURNS [BufferDefs.Buffer] = INLINE
  BEGIN RETURN[BufferDefs.Dequeue[@globalOutputQueue]]; END;

GrabInputBuffer: ENTRY PROCEDURE RETURNS [BufferDefs.Buffer] = INLINE
  BEGIN RETURN[BufferDefs.Dequeue[@globalInputQueue]]; END;

Wait: ENTRY PROCEDURE = INLINE
  BEGIN
  IF BufferDefs.QueueLength[@globalInputQueue] = 0 AND BufferDefs.QueueLength[
    @globalOutputQueue] = 0 THEN WAIT dispatcherReady;
  END;

SendToNextNetwork: PROCEDURE [b: BufferDefs.Buffer] =
  BEGIN
  network: DriverDefs.Network ← b.network;
  b.network ← network ← network.next;
  -- dangerous we are playing with monitor data
  IF network = NIL THEN
    BEGIN
    b.allNets ← FALSE; -- this is where it gets turned off
    b.requeueProcedure[b];
    RETURN;
    END;
  SELECT b.type FROM
    pup, rpp =>
      BEGIN
      pupRouter.broadcast[b]; -- this requeues b

      END;
    ois =>
      BEGIN
      oisRouter.broadcast[b]; -- this requeues b

      END;
    ENDCASE => b.requeueProcedure[b]; -- don't know how to encapsulate

  END;

-- Locked procedures
-- These are called from interrupt routines on the Alto; they must be locked in memory.

PutOnGlobalInputQueue: PUBLIC ENTRY PROCEDURE [b: BufferDefs.Buffer] =
  BEGIN BufferDefs.Enqueue[@globalInputQueue, b]; NOTIFY dispatcherReady; END;

PutOnGlobalDoneQueue: PUBLIC PROCEDURE [b: BufferDefs.Buffer] =
  BEGIN
  -- This test avoids an unnecessary process switch in returning buffers to system pool.
  IF ~b.allNets AND b.requeueProcedure = DriverDefs.ReturnFreeBuffer THEN
    DriverDefs.ReturnFreeBuffer[b]
  ELSE PutOnGlobalDoneQueueLocked[b];
  END;

PutOnGlobalDoneQueueLocked: ENTRY PROCEDURE [b: BufferDefs.Buffer] = INLINE
  BEGIN BufferDefs.Enqueue[@globalOutputQueue, b]; NOTIFY dispatcherReady; END;

-- initialization
```



```
  pupRouter ← @dummyRouterObject;
  oisRouter ← @dummyRouterObject;
  dummyRouter ← @dummyRouterObject;
  Process.SetTimeout[@dispatcherReady, Process.MsecToTicks[30000]];
  IF DriverDefs.doDebug THEN
    BEGIN
    DriverDefs.GetGiantVector[].globalInputQueue ← @globalInputQueue;
    DriverDefs.GetGiantVector[].globalOutputQueue ← @globalOutputQueue;
    END;

END. -- DispatcherImpl module
```

LOG

Time: April 21, 1980 2:56 PM By: Dalal Action: created file for Pilot 5.0.
Time: April 21, 1980 2:57 PM By: Dalal Action: merged the dispatchers.
*Time: May 7, 1980 10:34 AM By: BLyon Action: Made pupRouter & oisRouter PUBLIC (they are u **sed by Boss).*
*Time: May 8, 1980 1:24 PM By: BLyon Action: Made PutOnGlobalDoneQueue more efficient (hop **efully it still works).*

```
-- EchoServerImpl.mesa (last edited by: BLyon on: September 29, 1980 9:46 AM
-- Function: The implementation module for the Pilot OISCP Echo Server.

DIRECTORY
  CommunicationInternal USING [],
  DriverDefs USING [doStats],
  Echo USING [echoRequest, echoResponse],
  Heap USING [FreeMDSNode, MakeMDSNode],
  Socket USING [
    Abort, ChannelAborted, ChannelError, ChannelHandle, Create, Delete, Get,
    GetStatus, PhysicalRecord, PhysicalRecordHandle, Put, SetWaitTime,
    SocketStatus, TimeOut, TransferStatus, TransferWaitAny, WaitTime],
  Router USING [FindMyHostID],
  StatsDefs USING [StatIncr, StatBump],
  OISCPTypes USING [
    bytesPerOisPktText, echoerSocket, OisAddress, OisPacketType, unknownNetID],
  Space USING [wordsPerPage],
  SpecialSystem USING [NetworkAddress],
  System USING [];

EchoServerImpl: PROGRAM
  IMPORTS Heap, Router, Socket, StatsDefs
  EXPORTS CommunicationInternal, Echo, System =
  BEGIN

  -- EXPORTED TYPE(s)
  NetworkAddress: PUBLIC TYPE = SpecialSystem.NetworkAddress;

  -- global constants to all
  nBuffers: CARDINAL = 2;
  physicalRecordByteSize: CARDINAL =
    (2*SIZE[Socket.PhysicalRecord]) + OISCPTypes.bytesPerOisPktText;
  physicalRecordWordSize: CARDINAL = (physicalRecordByteSize + 1)/2;
  nWordsNeeded: CARDINAL = physicalRecordWordSize*nBuffers;
  nPagesNeeded: CARDINAL = (nWordsNeeded/Space.wordsPerPage) + 1;

  -- global variables to all
  cH: Socket.ChannelHandle;
  localAddr: OISCPTypes.OisAddress;
  echoServerFork: PROCESS;
  echoerTimeout: Socket.WaitTime = 7777777777B; -- in msec (about 10 days)

  -- Cold Procedures
  -- This procedure creates an echo server.
  CreateServer: PUBLIC PROCEDURE =
    BEGIN
    localAddr ←
      [net: OISCPTypes.unknownNetID, host: Router.FindMyHostID[],
        socket: OISCPTypes.echoerSocket];
    cH ← Socket.Create[localAddr];
    echoServerFork ← FORK EchoServer[];
    END; -- CreateServer

  -- This procedure deletes the echo server.

  DeleteServer: PUBLIC PROCEDURE =
    BEGIN Socket.Abort[cH]; JOIN echoServerFork; Socket.Delete[cH]; END;
  -- DeleteListener
```



```
  -- Hot Procedures
  -- This procedure echos Echo protocol packets back to the remote end.

  EchoServer: PROCEDURE =
    BEGIN
    personalBufferPool: POINTER;
    buf: Socket.PhysicalRecordHandle;
    i: CARDINAL;
    ok: BOOLEAN ← TRUE;
    socketStatus: Socket.SocketStatus;
    status: Socket.TransferStatus;
    Socket.SetWaitTime[cH, echoerTimeout];
    -- create the buffers that we need for receiving packets
    personalBufferPool ← Heap.MakeMDSNode[n: nWordsNeeded];
    -- init the PhysicalRecords and start the gets
    buf ← personalBufferPool;
    FOR i IN [0..nBuffers) DO
      buf.header.pktLength ← physicalRecordByteSize;
      [] ← Socket.Get[
        cH, buf ! Socket.ChannelError => BEGIN ok ← FALSE; EXIT; END;
        Socket.ChannelAborted => BEGIN ok ← FALSE; EXIT; END];
      buf ← buf + physicalRecordWordSize;
      ENDLOOP;
    IF ok THEN
      DO
        -- wait for an echo request packet
        [buf, status] ← Socket.TransferWaitAny[
          cH ! Socket.TimeOut => RETRY; Socket.ChannelAborted => EXIT];
        -- b.status can be one of aborted or goodCompletion
        SELECT status FROM
          goodCompletion =>
            BEGIN
            IF LOOPHOLE[buf.header.packetType, OISCPTypes.OisPacketType] = echo
              THEN
              BEGIN
              firstWord: POINTER TO CARDINAL ← LOOPHOLE[@buf.body, POINTER];
              -- examine this packet
              IF firstWord↑ = Echo.echoRequest THEN
                BEGIN
                -- good packet, echo it
                buf.header.destination ← buf.header.source;
                firstWord↑ ← Echo.echoResponse;
                [] ← Socket.Put[cH, buf ! Socket.ChannelAborted => EXIT];
                IF DriverDefs.doStats THEN StatsDefs.StatIncr[packetsEchoed];
                IF DriverDefs.doStats THEN
                  StatsDefs.StatBump[bytesEchoed, buf.header.pktLength];
                END
              ELSE IF DriverDefs.doStats THEN StatsDefs.StatIncr[packetsBadEchoed]
              END
            ELSE IF DriverDefs.doStats THEN StatsDefs.StatIncr[packetsBadEchoed];
            END; -- end goodCompletion

          aborted => EXIT;
          ENDCASE => NULL;
        buf.header.pktLength ← physicalRecordByteSize;
        [] ← Socket.Get[
          cH, buf ! Socket.ChannelError => EXIT; Socket.ChannelAborted => EXIT];
        ENDLOOP;
```

```
    -- we got here via an abort or error; now clean up this world.
    Socket.Abort[cH];
    -- clear the get queue
    socketStatus ← Socket.GetStatus[cH];
    FOR i IN [0..socketStatus.incompleteGets) DO
      [buf, status] ← Socket.TransferWaitAny[cH]; ENDLOOP;
    Heap.FreeMDSNode[p: personalBufferPool];
    END; -- EchoServer

  END.
```

LOG

Time: January 31, 1980 5:18 PM By: Dalal Action: create file.
Time: August 27, 1980 11:11 AM By: BLyon Action: increased timeOut.
Time: September 18, 1980 4:32 PM By: BLyon Action: Added StatsDefs.
Time: September 26, 1980 11:28 AM By: BLyon Action: made it use Physical records instead of lo
***cked buffers.*

```
-- EthernetDriver.mesa (last edited by: BLyon on: September 29, 1980 2:13 PM

DIRECTORY
  BufferDefs,
  CommUtilDefs USING [Zero, GetEthernetHostNumber],
  DriverDefs USING [
    doDebug, doStats, Glitch, GetInputBuffer, MaybeGetFreeBuffer, NetworkObject,
    AddDeviceToChain, PutOnGlobalDoneQueue, PutOnGlobalInputQueue],
  DriverTypes USING [Encapsulation, XWirePacketType],
  EthernetFace,
  Heap USING [MakeNode, FreeNode],
  Inline USING [LowHalf, HighHalf, BITAND, LongCOPY],
  OISCPTypes USING [allHostIDs],
  PupTypes USING [allHosts, PupErrorCode, PupHostID],
  StatsDefs USING [StatBump, StatIncr, StatCounterIndex],
  PilotSwitches USING [switches],
  Process USING [
    DisableTimeout, MsecToTicks, SecondsToTicks, SetPriority, SetTimeout, Ticks],
  ProcessInternal USING [AllocateNakedCondition, DeallocateNakedCondition],
  Runtime USING [SelfDestruct],
  SpecialSystem USING [
    ProcessorID, HostNumber, GetProcessorID, broadcastHostNumber, nullHostNumber,
    nullNetworkNumber],
  System USING [
    GetGreenwichMeanTime, GetClockPulses, Microseconds, Pulses,
    MicrosecondsToPulses, GreenwichMeanTime];

EthernetDriver: MONITOR
  IMPORTS
    BufferDefs, CommUtilDefs, DriverDefs, StatsDefs, EthernetFace, Heap, Inline,
    PilotSwitches, Process, ProcessInternal, Runtime, System, SpecialSystem
  EXPORTS DriverDefs
  SHARES BufferDefs, DriverTypes, SpecialSystem =
  BEGIN OPEN StatsDefs, BufferDefs, DriverDefs, EthernetFace, SpecialSystem;

  -- Things that should be in DriverTypes:
  translationPacket: DriverTypes.XWirePacketType = LOOPHOLE[1001B];
  xwireBroadcastHost: SpecialSystem.HostNumber =
    SpecialSystem.broadcastHostNumber;

  ether: DeviceHandle;
  me: SpecialSystem.ProcessorID;
  globalStatePtr: GlobalStatePtr; -- Allocate space if needed
  inProcess, outProcess: PROCESS;
  inWait, outWait: LONG POINTER TO CONDITION ← NIL;
  firstOutputBuffer, lastOutputBuffer: Buffer;
  firstInputBuffer, lastInputBuffer: Buffer;
  inInterruptMask, outInterruptMask: WORD;

  watcherProcess: PROCESS;
  pleaseStop: BOOLEAN;
  timer: CONDITION;
  timeLastRecv, timeSendStarted: System.Pulses;
  fiveSecondsOfPulses: System.Pulses;
  fiveHalfSecondsOfPulses: System.Pulses;

  inputQueueLength: CARDINAL ← 4;
```



```
  myNetwork: DriverDefs.NetworkObject ←
    [decapsulateBuffer: DecapsulateBuffer, encapsulatePup: EncapsulatePup,
      encapsulateRpp: EncapsulateRpp, encapsulateOis: EncapsulateOis,
      sendBuffer: SendBuffer, forwardBuffer: ForwardBuffer,
      activateDriver: ActivateDriver, deactivateDriver: DeactivateDriver,
      deleteDriver: DeleteDriver, alive: TRUE, speed: 10000, -- in kiloBits/sec
      buffers:, spare:, interrupt: InInterrupt, device: xwire, index:,
      netNumber: nullNetworkNumber, hostNumber:, next: NIL, pupStats: NIL,
      stats: NIL];

  FunnyRetransmissionMask: PUBLIC ERROR = CODE;
  MachineIDTooBigForEthernet: PUBLIC ERROR = CODE;
  DriverNotActive: PUBLIC ERROR = CODE;
  DriverAlreadyActive: PUBLIC ERROR = CODE;
  EthernetNetNumberScrambled: PUBLIC ERROR = CODE;
  CantMakImageWhileEtherentDriverIsActive: PUBLIC ERROR = CODE;
  OnlyTwoDriversArePossible: PUBLIC ERROR = CODE;
  BufferMustBeAlmostQuadWordAligned: PUBLIC ERROR = CODE;
  IOCBMustBeQuadWordAligned: PUBLIC ERROR = CODE;
  IOCBMustBeInFirstMDS: PUBLIC ERROR = CODE;

  EtherStatsInfo: TYPE = RECORD [
    packetsSent: LONG CARDINAL,
    badSendSatus: LONG CARDINAL,
    overruns: LONG CARDINAL,
    packetsRecv: LONG CARDINAL,
    badRecvStatus: LONG CARDINAL,
    inputOff: LONG CARDINAL,
    loadTable: ARRAY [0..16] OF LONG CARDINAL];
  etherStatsInfo: EtherStatsInfo;
  etherStats: POINTER TO EtherStatsInfo ← @etherStatsInfo;

  -- Hot Procedures

  InInterrupt: ENTRY PROCEDURE =
    BEGIN
    this, new: Buffer;
    status: Status;
    lastMissed, missed: CARDINAL ← GetPacketsMissed[ether];

    Process.SetPriority[5];

    UNTIL pleaseStop DO
      this ← firstInputBuffer;
      status ← GetStatus[this.iocbChain];
      IF doStats AND status # pending THEN
        StatIncr[statEtherInterruptDuringInterrupt];
      UNTIL pleaseStop OR status # pending DO
        WAIT inWait;
        status ← GetStatus[this.iocbChain];
        IF doStats AND status = pending THEN StatIncr[statEtherMissingStatus];
        ENDLOOP;
      IF doStats AND (missed ← GetPacketsMissed[ether]) # lastMissed THEN
        BEGIN
        StatBump[statEtherEmptyNoBuffer, missed - lastMissed];
        lastMissed ← missed;
        END;
      IF pleaseStop THEN EXIT;
```

```
    firstInputBuffer ← firstInputBuffer.next;
    SELECT status FROM
      ok =>
        BEGIN
        timeLastRecv ← System.GetClockPulses[];
        IF (new ← GetInputBuffer[]) # NIL THEN
          BEGIN
          new.device ← xwire;
          this.length ← GetPacketLength[this.iocbChain];
          this.network ← @myNetwork;
          IF doStats THEN
            BEGIN
            etherStats.packetsRecv ← etherStats.packetsRecv + 1;
            StatIncr[statEtherPacketsReceived];
            StatBump[statEtherWordsReceived, this.length];
            END;
          PutOnGlobalInputQueue[this];
          END
        ELSE
          BEGIN
          new ← this; -- Rats, couldn't get a new buffer, recycle this one
          IF doStats THEN
            BEGIN
            etherStats.inputOff ← etherStats.inputOff + 1;
            StatIncr[statEtherEmptyFreeQueue];
            END;
          END;
        END;
      ENDCASE =>
        BEGIN
        new ← this; -- Some kind of error, recycle this buffer
        IF doStats THEN
          SELECT status FROM
            packetTooLong => StatIncr[statEtherReceivedTooLong];
            badAlignmentButOkCrc => StatIncr[statEtherReceivedNot16];
            crc => StatIncr[statEtherReceivedBadCRC];
            crcAndBadAlignment => StatIncr[statEtherReceivedNot16BadCRC];
            overrun =>
              BEGIN
              etherStats.overruns ← etherStats.overruns + 1;
              StatIncr[statEtherReceivedOverrun];
              END;
            ENDCASE => StatIncr[statEtherReceivedBadStatus];
        END;
    -- add new buffer to end of input chain
    QueueInput[ether, @new.encapsulation, new.length, new.iocbChain];
    lastInputBuffer.next ← new;
    lastInputBuffer ← new;
    new.next ← NIL;
    ENDLOOP;
  END;

OutInterrupt: ENTRY PROCEDURE =
  BEGIN
  b: Buffer;
  status: Status;

  Process.SetPriority[5];

  UNTIL pleaseStop DO
```



```
    DO
      -- forever until something interesting happens
      IF pleaseStop THEN EXIT;
      -- we compute the values each time around since the value of b can change if
      -- the watcher shoots down the output.
      IF (b ← firstOutputBuffer) # NIL AND (status ← GetStatus[b.iocbChain]) #
        pending THEN EXIT;
      WAIT outWait;
      ENDLOOP;
    IF pleaseStop THEN EXIT; -- so that we do not do something below

    SELECT status FROM
      ok =>
        BEGIN
        IF doStats THEN
          BEGIN
          tries: CARDINAL ← GetRetries[b.iocbChain];
          statEtherSendsCollision1: StatsDefs.StatCounterIndex =
            statEtherSendsCollision1;
          first: CARDINAL = LOOPHOLE[statEtherSendsCollision1];
          etherStats.packetsSent ← etherStats.packetsSent + 1;
          IF tries # 0 THEN StatIncr[LOOPHOLE[first + tries]];
          etherStats.loadTable[tries] ← etherStats.loadTable[tries] + 1;
          END;
        END;
      ENDCASE =>
        BEGIN
        IF doStats THEN
          SELECT status FROM
            tooManyCollisions =>
              BEGIN
              etherStats.loadTable[16] ← etherStats.loadTable[16] + 1;
              StatIncr[statEtherSendsCollisionLoadOverflow];
              END;
            underrun =>
              BEGIN
              etherStats.overruns ← etherStats.overruns + 1;
              StatIncr[statEtherSendOverrun];
              END;
            ENDCASE =>
              BEGIN
              etherStats.badSendSatus ← etherStats.badSendSatus + 1;
              StatIncr[statEtherSendBadStatus];
              END;
        END;

    -- We don't resend things that screwup
    firstOutputBuffer ← firstOutputBuffer.next;
    PutOnGlobalDoneQueue[b];
    ENDLOOP;
  END;

GetElapsedPulses: PROCEDURE [startTime: System.Pulses] RETURNS [System.Pulses] =
  INLINE
  BEGIN
  RETURN[LOOPHOLE[System.GetClockPulses[] - startTime, System.Pulses]];
  END;

Watcher: PROCEDURE =
  BEGIN
```

```
  UNTIL pleaseStop DO
    -- In either case, an interrupt should be pending.  Since the interrupt routine is higher priority
**than we are, it should get processed before we can see it.  If we get here, an interrupt has proba
**bly been lost.  It could have been generated between the time we started decoding the instructi
**on and the time that the data is actually fetched.  That is why we look several times.  Of course, i
**f it is still not zero when we look again, it could be a new interrupt that has just arrived.
    -- Check for lost input interrupt
    THROUGH [0..25) DO
      IF InputChainOK[] THEN EXIT;
      REPEAT FINISHED => BEGIN WatcherNotify[]; END;
      ENDLOOP;
    -- Check for lost output interrupt
    THROUGH [0..25) DO
      IF OutputChainOK[] THEN EXIT;
      REPEAT FINISHED => BEGIN WatcherNotify[]; END;
      ENDLOOP;
    -- Check for stuck input
    IF GetElapsedPulses[timeLastRecv] > fiveSecondsOfPulses THEN FixupInput[];
    -- Check for stuck output
    IF firstOutputBuffer # NIL AND
      (GetElapsedPulses[timeSendStarted] > fiveHalfSecondsOfPulses) THEN
      ShootDownOutput[];
    WatcherWait[];
    ENDLOOP;
  END;

InputChainOK: ENTRY PROCEDURE RETURNS [BOOLEAN] = INLINE
  BEGIN RETURN[GetStatus[firstInputBuffer.iocbChain] = pending]; END;

OutputChainOK: ENTRY PROCEDURE RETURNS [BOOLEAN] = INLINE
  BEGIN
  RETURN[
    (firstOutputBuffer = NIL) OR
      (GetStatus[firstOutputBuffer.iocbChain] = pending)];
  END;

WatcherWait: ENTRY PROCEDURE = INLINE BEGIN WAIT timer; END;

WatcherNotify: ENTRY PROCEDURE = INLINE
  BEGIN
  IF doStats THEN StatIncr[statEtherLostInterrupts];
  NOTIFY inWait↑;
  NOTIFY outWait↑;
  END;

FixupInput: ENTRY PROCEDURE = INLINE
  BEGIN
  IF doStats THEN StatIncr[statInputIdle];
  SmashCSBs[]; -- this will leave output dangling

  END;

ShootDownOutput: ENTRY PROCEDURE = INLINE
  BEGIN
  -- This happens if the transciever is unplugged
  b: Buffer;
  TurnOff[ether];
  UNTIL firstOutputBuffer = NIL DO
```



```
    b ← firstOutputBuffer;
    firstOutputBuffer ← firstOutputBuffer.next;
    PutOnGlobalDoneQueue[b];
    IF doStats THEN StatIncr[statPacketsStuckInOutput];
    ENDLOOP;
  SmashCSBs[];
  END;

DecapsulateBuffer: PROCEDURE [b: Buffer] RETURNS [BufferType] =
  BEGIN
  SELECT b.encapsulation.xwireType FROM
    pup =>
      BEGIN
      IF 2*b.length < b.pupLength + 2*SIZE[DriverTypes.Encapsulation] THEN
        BEGIN
        IF doStats THEN StatIncr[statPupsDiscarded];
        RETURN[rejected];
        END;
      RETURN[pup];
      END;
    ois =>
      BEGIN
      IF 2*b.length < b.oisPktLength + 2*SIZE[DriverTypes.Encapsulation] THEN
        BEGIN IF doStats THEN StatIncr[statOisDiscarded]; RETURN[rejected]; END;
      RETURN[ois];
      END;
    translationPacket =>
      BEGIN
      IF b.rawWords[0] = translationRequest THEN ReceiveRequest[b]
      ELSE IF b.rawWords[0] = translationResponse THEN ReceiveAck[b];
      RETURN[rejected];
      END;
    ENDCASE => RETURN[rejected];
  END;

EncapsulateRpp: PROCEDURE [RppBuffer, PupHostID] = LOOPHOLE[EncapsulatePup];
EncapsulatePup: PROCEDURE [b: PupBuffer, destination: PupHostID] =
  BEGIN
  foundIt: BOOLEAN;
  oisAddr: OisAddr;
  [foundIt, oisAddr] ← Translate[destination];
  IF foundIt THEN
    BEGIN
    b.encapsulation ←
      [xwire[xwireDest: oisAddr, xwireSource: me, xwireType: pup]];
    b.length ← (b.pupLength + 1)/2 + SIZE[DriverTypes.Encapsulation];
    END
  ELSE
    BEGIN
    b.encapsulation ←
      [xwire[
        xwireDest:, xwireSource: SpecialSystem.nullHostNumber,
        -- Marker for translation failed
        xwireType: pup]];
    END;
  END;

EncapsulateOis: PROCEDURE [
  b: OisBuffer, destination: SpecialSystem.HostNumber] =
  BEGIN
```

```
    b.encapsulation ←
      [xwire[xwireDest: destination, xwireSource: me, xwireType: ois]];
    b.length ← (b.oisPktLength + 1)/2 + SIZE[DriverTypes.Encapsulation];
    END;

  ForwardBuffer: PROCEDURE [b: Buffer] RETURNS [PupTypes.PupErrorCode] =
    BEGIN
    IF FALSE THEN -- outputQueue.length>10 THEN
      RETURN[gatewayResourceLimitsPupErrorCode]; -- transceiver unplugged?
    SendBuffer[b];
    RETURN[noErrorPupErrorCode];
    END;

  SendBuffer: ENTRY PROCEDURE [b: Buffer] =
    BEGIN
    IF pleaseStop THEN Glitch[DriverNotActive];
    IF b.encapsulation.xwireSource = SpecialSystem.nullHostNumber THEN
      BEGIN PutOnGlobalDoneQueue[b]; RETURN; END;
    IF ~hearSelf AND
      (b.encapsulation.xwireDest = me OR b.encapsulation.xwireDest =
        xwireBroadcastHost) THEN
      BEGIN -- sending to ourself, copy it over since we can't hear it
      copy: Buffer ← GetInputBuffer[];
      IF copy # NIL THEN
        BEGIN
        copy.device ← xwire;
        Inline.LongCOPY[
          from: @b.encapsulation, nwords: b.length, to: @copy.encapsulation];
        copy.length ← b.length;
        copy.network ← @myNetwork;
        IF doStats THEN StatIncr[statEtherPacketsLocal];
        IF doStats THEN StatBump[statEtherWordsLocal, b.length];
        PutOnGlobalInputQueue[copy];
        END
      ELSE IF doStats THEN StatIncr[statEtherEmptyFreeQueue];
      END;
    SendBufferInternal[b];
    END;

  SendBufferInternal: INTERNAL PROCEDURE [b: Buffer] =
    BEGIN
    b.device ← xwire;
    QueueOutput[ether, @b.encapsulation, b.length, b.iocbChain];
    IF doStats THEN StatIncr[statEtherPacketsSent];
    IF doStats THEN StatBump[statEtherWordsSent, b.length];
    IF firstOutputBuffer = NIL THEN firstOutputBuffer ← b
    ELSE lastOutputBuffer.next ← b;
    lastOutputBuffer ← b;
    timeSendStarted ← System.GetClockPulses[];
    END;

  -- COLD code, only used when turning things on + off

  AdjustLengtoOfD0EthernetInputQueue: PUBLIC PROCEDURE [n: CARDINAL] =
    BEGIN inputQueueLength ← n; END;

  CreateDefaultEthernetDrivers: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
    BEGIN
    deviceNumber: CARDINAL ← 0;
```



```
    etherDevice: DeviceHandle ← GetNextDevice[nullDeviceHandle];
    IF PilotSwitches.switches.b = down THEN RETURN[FALSE];
    IF etherDevice = nullDeviceHandle THEN RETURN[FALSE];
    WHILE etherDevice # nullDeviceHandle DO
      CreateAnEthernetDriver[etherDevice, deviceNumber];
      etherDevice ← GetNextDevice[etherDevice];
      deviceNumber ← deviceNumber + 1;
      ENDLOOP;
    RETURN[TRUE];
    END;

  CreateAnEthernetDriver: PROCEDURE [
    etherDevice: DeviceHandle, deviceNumber: CARDINAL] =
    BEGIN
    IF deviceNumber # 0 THEN
      BEGIN
      him: POINTER TO FRAME[EthernetDriver] ← NEW EthernetDriver;
      him.SetupEthernetDriver[etherDevice];
      END
    ELSE SetupEthernetDriver[etherDevice];
    END;

  SetupEthernetDriver: PROCEDURE [etherDevice: DeviceHandle] =
    BEGIN
    ether ← etherDevice;
    me ← SpecialSystem.GetProcessorID[];
    myNetwork.netNumber ← nullNetworkNumber;
    pleaseStop ← TRUE;
    myNetwork.buffers ← inputQueueLength;
    AddDeviceToChain[@myNetwork, controlBlockSize];
    IF doStats THEN
      BEGIN
      myNetwork.stats ← etherStats;
      CommUtilDefs.Zero[etherStats, SIZE[EtherStatsInfo]];
      END;
    END;

  ActivateDriver: PROCEDURE =
    BEGIN
    b: Buffer;
    IF ~pleaseStop THEN Glitch[DriverAlreadyActive];
    fiveSecondsOfPulses ← System.MicrosecondsToPulses[5000000];
    fiveHalfSecondsOfPulses ← System.MicrosecondsToPulses[2500000];
    pleaseStop ← FALSE;
    TurnOff[ether];
    AddCleanup[ether];
    firstInputBuffer ← lastInputBuffer ← NIL;
    firstOutputBuffer ← lastOutputBuffer ← NIL;
    myNetwork.hostNumber ← CommUtilDefs.GetEthernetHostNumber[];
    THROUGH [0..inputQueueLength) DO
      b ← GetInputBuffer[];
      IF doDebug AND Inline.BITAND[Inline.LowHalf[@b.encapsulation], 3] # 3 THEN
        Glitch[BufferMustBeAlmostQuadWordAligned];
      IF doDebug AND Inline.BITAND[Inline.LowHalf[b.iocbChain], 3] # 0 THEN
        Glitch[IOCBMustBeQuadWordAligned];
      IF doDebug AND Inline.HighHalf[b.iocbChain] # 0 THEN
        Glitch[IOCBMustBeInFirstMDS];
      IF firstInputBuffer = NIL THEN firstInputBuffer ← b;
      IF lastInputBuffer # NIL THEN lastInputBuffer.next ← b;
```

```
    lastInputBuffer ← b;
    ENDLOOP;
  [cv: inWait, mask: inInterruptMask] ← ProcessInternal.AllocateNakedCondition[
    ];
  Process.DisableTimeout[inWait];
  [cv: outWait, mask: outInterruptMask] ←
    ProcessInternal.AllocateNakedCondition[];
  Process.DisableTimeout[outWait];
  SmashCSBs[];
  inProcess ← FORK InInterrupt[];
  outProcess ← FORK OutInterrupt[];
  watcherProcess ← FORK Watcher[];
  CreateCache[];
  END;

SmashCSBs: PROCEDURE =
  BEGIN
  b: Buffer;
  TurnOff[ether];
  TurnOn[ether, me, inInterruptMask, outInterruptMask, globalStatePtr];
  FOR b ← firstInputBuffer, b.next UNTIL b = NIL DO
    QueueInput[ether, @b.encapsulation, b.length, b.iocbChain]; ENDLOOP;
  timeLastRecv ← System.GetClockPulses[];
  END;

DeleteDriver: PROCEDURE =
  BEGIN
  IF ether # GetNextDevice[nullDeviceHandle] THEN Runtime.SelfDestruct[];
  END;

DeactivateDriver: PROCEDURE =
  BEGIN
  b: Buffer;
  IF pleaseStop THEN Glitch[DriverNotActive];
  pleaseStop ← TRUE;
  KillInterruptRoutines[];
  JOIN inProcess;
  JOIN outProcess;
  TurnOff[ether];
  ProcessInternal.DeallocateNakedCondition[inWait];
  ProcessInternal.DeallocateNakedCondition[outWait];
  inWait ← outWait ← NIL;
  KillDriverLocked[];
  JOIN watcherProcess;
  RemoveCleanup[ether];
  UNTIL firstInputBuffer = NIL DO
    b ← firstInputBuffer;
    firstInputBuffer ← b.next;
    ReturnFreeBuffer[b];
    ENDLOOP;
  UNTIL firstOutputBuffer = NIL DO
    b ← firstOutputBuffer;
    firstOutputBuffer ← b.next;
    ReturnFreeBuffer[b];
    ENDLOOP;
  myNetwork.netNumber ← nullNetworkNumber;
  -- in case we turn it on after moving to another machine
  DeleteCache[];
```



```
  END;

KillInterruptRoutines: ENTRY PROCEDURE = INLINE
  BEGIN NOTIFY inWait↑; NOTIFY outWait↑; END;

KillDriverLocked: ENTRY PROCEDURE = INLINE BEGIN NOTIFY timer; END;

OisAddr: TYPE = SpecialSystem.HostNumber;
Ethernet1Addr: TYPE = PupTypes.PupHostID;
AddressPair: TYPE = MACHINE DEPENDENT RECORD [
  oisAddr: OisAddr, ethernet1Addr: Ethernet1Addr, filler: [0..377B]];

CacheEntry: TYPE = LONG POINTER TO CacheObject;

CacheObject: TYPE = MACHINE DEPENDENT RECORD [
  nextLink: CacheEntry,
  addressPair: AddressPair,
  tries: CARDINAL,
  timeStamp: System.GreenwichMeanTime,
  status: CacheStatus,
  filler: [0..37777B]];

CacheStatus: TYPE = {new, pending, active, zombie};

-- variables
translationRequest: CARDINAL = 10101B;
translationResponse: CARDINAL = 7070B;
cacheQueueHead: CacheEntry;
broadCastPairEntry: CacheEntry; -- permanent
myAddressPairEntry: CacheEntry; -- permanent
retryLimit: CARDINAL ← 10B;
retryTime: LONG CARDINAL ← 2; -- two seconds
demonActiveTime: Process.Ticks ← Process.SecondsToTicks[1]; -- one second
deactivateTime: LONG CARDINAL ← 3*60; -- three minutes
demonSleepTime: Process.Ticks ← Process.SecondsToTicks[60*5]; -- five minutes
cacheEvent: CONDITION;
demonProcess: PROCESS;


-- interface
CreateCache: ENTRY PROCEDURE = INLINE
  BEGIN
  aP: AddressPair ← [OISCPTypes.allHostIDs, PupTypes.allHosts, 0];
  Process.SetTimeout[@cacheEvent, demonSleepTime];
  cacheQueueHead ← NIL;
  broadCastPairEntry ← AddAddressPair[aP];
  aP ← [me, [myNetwork.hostNumber], 0];
  myAddressPairEntry ← AddAddressPair[aP];
  demonProcess ← FORK Demon[];
  END;

DeleteCache: PROCEDURE = INLINE
  BEGIN
  e, nextE: CacheEntry;

  DeleteCacheLocked: ENTRY PROCEDURE = INLINE BEGIN NOTIFY cacheEvent; END;
  -- DeleteCacheLocked
```

```
  DeleteCacheLocked[];
  JOIN demonProcess;
  e ← cacheQueueHead;
  cacheQueueHead ← NIL;
  WHILE e # NIL DO nextE ← e.nextLink; Heap.FreeNode[p: e]; e ← nextE; ENDLOOP;
  END;

depth: CARDINAL;
FindEntry: INTERNAL PROCEDURE [ethernet1Addr: Ethernet1Addr]
  RETURNS [entry: CacheEntry] =
  BEGIN
  IF doStats THEN depth ← 0;
  entry ← cacheQueueHead;
  WHILE entry # NIL DO
    IF ethernet1Addr = entry.addressPair.ethernet1Addr THEN RETURN;
    entry ← entry.nextLink;
    IF doStats THEN depth ← depth + 1;
    ENDLOOP;
  END;

AddEntry: INTERNAL PROCEDURE [entry: CacheEntry] =
  BEGIN entry.nextLink ← cacheQueueHead; cacheQueueHead ← entry; END;

RemoveEntry: INTERNAL PROCEDURE [entry: CacheEntry] =
  BEGIN
  e, pred: CacheEntry;
  IF (pred ← cacheQueueHead) = entry THEN
    BEGIN cacheQueueHead ← cacheQueueHead.nextLink; RETURN; END;
  e ← pred.nextLink;
  WHILE e # NIL DO
    IF e = entry THEN BEGIN pred.nextLink ← entry.nextLink; RETURN; END;
    pred ← e;
    e ← pred.nextLink;
    ENDLOOP;
  ERROR; -- entry not found

  END;


Translate: ENTRY PROCEDURE [ethernet1Addr: Ethernet1Addr]
  RETURNS [foundIt: BOOLEAN, oisAddr: OisAddr] =
  BEGIN
  e: CacheEntry;
  IF (e ← FindEntry[ethernet1Addr]) # NIL THEN
    BEGIN
    IF foundIt ← (e.status = active) THEN
      BEGIN
      oisAddr ← e.addressPair.oisAddr;
      e.timeStamp ← System.GetGreenwichMeanTime[];
      END;
    IF e # cacheQueueHead THEN -- put e at the head of the queue
      BEGIN
      IF doStats THEN StatBump[cacheDepth, depth];
      RemoveEntry[e];
      AddEntry[e];
      END;
    END
  ELSE -- entry not found, so add a new one
    BEGIN
```



```
    foundIt ← FALSE;
    IF doStats THEN StatIncr[cacheFault];
    e ← Heap.MakeNode[n: SIZE[CacheObject]];
    e.status ← new;
    e.tries ← 0;
    e.timeStamp ← System.GetGreenwichMeanTime[];
    e.addressPair ← [oisAddr:, ethernet1Addr: ethernet1Addr, filler:];
    AddEntry[e];
    NOTIFY cacheEvent;
    END;
  END;

AddAddressPair: INTERNAL PROCEDURE [aP: AddressPair] RETURNS [e: CacheEntry] =
  BEGIN
  IF (e ← FindEntry[aP.ethernet1Addr]) = NIL THEN
    BEGIN e ← Heap.MakeNode[n: SIZE[CacheObject]]; AddEntry[e]; END;
  e.addressPair ← aP;
  e.status ← active;
  e.timeStamp ← System.GetGreenwichMeanTime[];
  END;

DeallocateEntry: INTERNAL PROCEDURE [e: CacheEntry] =
  BEGIN
  -- there are two entries that we do not want to throw out!!
  IF (e = broadCastPairEntry) OR (e = myAddressPairEntry) THEN
    e.timeStamp ← System.GetGreenwichMeanTime[]
  ELSE BEGIN RemoveEntry[e]; Heap.FreeNode[p: e]; END;
  END;

Demon: ENTRY PROCEDURE =
  BEGIN
  t: LONG CARDINAL;
  e, nextE: CacheEntry;
  pendingEntries: BOOLEAN;
  Process.SetPriority[5];

  UNTIL pleaseStop DO
    WAIT cacheEvent;
    IF pleaseStop THEN EXIT;
    pendingEntries ← FALSE;
    e ← cacheQueueHead;
    WHILE (e # NIL) DO
      nextE ← e.nextLink;
      t ← System.GetGreenwichMeanTime[] - e.timeStamp;
      SELECT e.status FROM
        active, zombie =>
          BEGIN IF t > deactivateTime THEN DeallocateEntry[e]; END;
        pending =>
          BEGIN
          pendingEntries ← TRUE;
          IF t > retryTime THEN
            BEGIN
            e.tries ← e.tries + 1;
            IF e.tries > retryLimit THEN
              BEGIN
              e.status ← zombie;
              IF doStats THEN StatIncr[unsuccessfulTranslation];
              END
            ELSE
```

```
            BEGIN
            IF doStats THEN StatIncr[translationRetries];
            SendRequest[e];
            e.timeStamp ← System.GetGreenwichMeanTime[];
            END;
          END;
        END;
      new =>
        BEGIN
        pendingEntries ← TRUE;
        SendRequest[e];
        e.status ← pending;
        e.timeStamp ← System.GetGreenwichMeanTime[];
        END;
      ENDCASE => ERROR;
    e ← nextE;
    ENDLOOP; -- end of queue entries loop
  IF pendingEntries THEN Process.SetTimeout[@cacheEvent, demonActiveTime]
  ELSE Process.SetTimeout[@cacheEvent, demonSleepTime];
  ENDLOOP; -- end of infinite loop

END;

SendRequest: INTERNAL PROCEDURE [e: CacheEntry] =
  BEGIN
  b: Buffer;
  request: LONG POINTER TO AddressPair;
  IF (b ← MaybeGetFreeBuffer[]) # NIL THEN
    BEGIN
    -- broadcast the trnslation request
    b.encapsulation ←
      [xwire[
        xwireDest: xwireBroadcastHost, xwireSource: me,
        xwireType: translationPacket]];
    b.length ← SIZE[DriverTypes.Encapsulation] + 2*SIZE[AddressPair] + 1;
    b.rawWords[0] ← translationRequest;
    request ← LOOPHOLE[@b.rawWords[1]];
    request↑ ← e.addressPair;
    -- also send our addresses, so responder does not fault
    request ← request + SIZE[AddressPair];
    request↑ ← myAddressPairEntry.addressPair;
    -- send it
    SendBufferInternal[b];
    END;
  END;

-- locks

ReceiveAck: ENTRY PROCEDURE [b: Buffer] = INLINE
  BEGIN
  IF b.encapsulation.xwireDest = myAddressPairEntry.addressPair.oisAddr THEN
    BEGIN
    receipt: LONG POINTER TO AddressPair ← LOOPHOLE[@b.rawWords[1]];
    [] ← AddAddressPair[receipt↑];
    END;
  END;

ReceiveRequest: ENTRY PROCEDURE [b: Buffer] = INLINE
  BEGIN
```



```
  request, requesterAddr: LONG POINTER TO AddressPair;
  request ← LOOPHOLE[@b.rawWords[1]];
  IF request.ethernet1Addr = myAddressPairEntry.addressPair.ethernet1Addr THEN
    BEGIN
    IF doStats THEN StatIncr[requestsForMe];
    -- since the requester is probably going to talk to us, add his address before we take a fault
    requesterAddr ← request + SIZE[AddressPair];
    [] ← AddAddressPair[requesterAddr↑];
    request.oisAddr ← myAddressPairEntry.addressPair.oisAddr;
    SendAck[request↑, b.encapsulation.xwireSource];
    END;
  END;

SendAck: INTERNAL PROCEDURE [aP: AddressPair, to: SpecialSystem.HostNumber] =
  INLINE
  BEGIN
  b: Buffer;
  response: LONG POINTER TO AddressPair;
  IF (b ← MaybeGetFreeBuffer[]) # NIL THEN
    BEGIN
    b.encapsulation ←
      [xwire[xwireDest: to, xwireSource: me, xwireType: translationPacket]];
    b.length ← SIZE[DriverTypes.Encapsulation] + SIZE[AddressPair] + 1;
    b.rawWords[0] ← translationResponse;
    response ← LOOPHOLE[@b.rawWords[1]];
    response↑ ← aP;
    -- send it
    SendBufferInternal[b];
    END;
  END;

-- initialization

Process.SetTimeout[@timer, Process.MsecToTicks[1000]];
END. -- EthernetDriver
```

September 5, 1980 1:36 AM By HGM; create from EthernetOneDriver.
September 17, 1980 4:41 PM By BLyon; added myNetwork.buffers stuff.

```
-- EthernetOneDriver.mesa (last edited by: BLyon on: September 29, 1980 2:05 PM)

DIRECTORY
  BufferDefs,
  CommUtilDefs USING [Zero],
  DriverDefs USING [
    GetGiantVector, GiantVector, doDebug, doStats, Glitch, GetInputBuffer,
    MaybeGetFreeBuffer, NetworkObject, AddDeviceToChain, PutOnGlobalDoneQueue,
    PutOnGlobalInputQueue],
  DriverTypes USING [
    Byte, ethernetEncapsulationOffset, ethernetEncapsulationBytes],
  Environment USING [Byte],
  EthernetOneFace,
  Heap USING [MakeNode, FreeNode],
  Inline USING [LowHalf, HighHalf, BITAND, LongCOPY],
  OISCPTypes USING [allHostIDs],
  PilotSwitches USING [switches],
  Process USING [
    DisableTimeout, MsecToTicks, SecondsToTicks, SetPriority, SetTimeout, Ticks],
  ProcessInternal USING [AllocateNakedCondition, DeallocateNakedCondition],
  PupTypes USING [allHosts, PupErrorCode, PupHostID],
  Runtime USING [SelfDestruct],
  StatsDefs USING [StatBump, StatIncr, StatCounterIndex],
  SpecialSystem USING [
    ProcessorID, HostNumber, GetProcessorID, nullNetworkNumber],
  System USING [
    GetGreenwichMeanTime, GetClockPulses, Microseconds, Pulses,
    MicrosecondsToPulses, GreenwichMeanTime];

EthernetOneDriver: MONITOR
  IMPORTS
    BufferDefs, CommUtilDefs, DriverDefs, StatsDefs, EthernetOneFace, Heap,
    Inline, PilotSwitches, Process, ProcessInternal, Runtime, System,
    SpecialSystem
  EXPORTS CommUtilDefs, DriverDefs
  SHARES BufferDefs, DriverTypes, SpecialSystem =
  BEGIN OPEN StatsDefs, BufferDefs, DriverDefs, EthernetOneFace, SpecialSystem;

  ethernetEncapsulationOffset: CARDINAL = DriverTypes.ethernetEncapsulationOffset;
  ethernetEncapsulationBytes: CARDINAL = DriverTypes.ethernetEncapsulationBytes;

  ether: DeviceHandle;
  globalStatePtr: GlobalStatePtr; -- Allocate space if needed
  inProcess, outProcess: PROCESS;
  inWait, outWait: LONG POINTER TO CONDITION ← NIL;
  firstOutputBuffer, lastOutputBuffer: Buffer;
  firstInputBuffer, lastInputBuffer: Buffer;
  inInterruptMask, outInterruptMask: WORD;

  watcherProcess: PROCESS;
  pleaseStop: BOOLEAN;
  timer: CONDITION;
  timeLastRecv, timeSendStarted: System.Pulses;
  fiveSecondsOfPulses: System.Pulses;
  fiveHalfSecondsOfPulses: System.Pulses;

  inputQueueLength: CARDINAL ← 4;
```



```
  myNetwork: DriverDefs.NetworkObject ←
    [decapsulateBuffer: DecapsulateBuffer, encapsulatePup: EncapsulatePup,
      encapsulateRpp: EncapsulateRpp, encapsulateOis: EncapsulateOis,
      sendBuffer: SendBuffer, forwardBuffer: ForwardBuffer,
      activateDriver: ActivateDriver, deactivateDriver: DeactivateDriver,
      deleteDriver: DeleteDriver, alive: TRUE, speed: 3000, -- in kiloBits/sec
      buffers:, spare:, interrupt: InInterrupt, device: ethernet, index:,
      netNumber: nullNetworkNumber, hostNumber:, next: NIL, pupStats: LOOPHOLE[0],
      -- Avoid binder complaints
      stats: NIL];

  FunnyRetransmissionMask: PUBLIC ERROR = CODE;
  MachineIDTooBigForEthernet: PUBLIC ERROR = CODE;
  DriverNotActive: PUBLIC ERROR = CODE;
  DriverAlreadyActive: PUBLIC ERROR = CODE;
  EthernetNetNumberScrambled: PUBLIC ERROR = CODE;
  CantMakImageWhileEtherentDriverIsActive: PUBLIC ERROR = CODE;
  OnlyTwoDriversArePossible: PUBLIC ERROR = CODE;
  BufferMustBeQuadWordAligned: PUBLIC ERROR = CODE;
  IOCBMustBeQuadWordAligned: PUBLIC ERROR = CODE;
  IOCBMustBeInFirstMDS: PUBLIC ERROR = CODE;

  EtherStatsInfo: TYPE = RECORD [
    packetsSent: LONG CARDINAL,
    badSendSatus: LONG CARDINAL,
    overruns: LONG CARDINAL,
    packetsRecv: LONG CARDINAL,
    badRecvStatus: LONG CARDINAL,
    inputOff: LONG CARDINAL,
    loadTable: ARRAY [0..16] OF LONG CARDINAL];
  etherStatsInfo: EtherStatsInfo;
  etherStats: POINTER TO EtherStatsInfo ← @etherStatsInfo;

  -- Hot Procedures

  InInterrupt: ENTRY PROCEDURE =
    BEGIN
    this, new: Buffer;
    status: Status;
    lastMissed, missed: CARDINAL ← GetPacketsMissed[ether];

    Process.SetPriority[5];

    UNTIL pleaseStop DO
      this ← firstInputBuffer;
      status ← GetStatus[this.iocbChain];
      IF doStats AND status # pending THEN
        StatIncr[statEtherInterruptDuringInterrupt];
      UNTIL pleaseStop OR status # pending DO
        WAIT inWait;
        status ← GetStatus[this.iocbChain];
        IF doStats AND status = pending THEN StatIncr[statEtherMissingStatus];
        ENDLOOP;
      IF doStats AND (missed ← GetPacketsMissed[ether]) # lastMissed THEN
        BEGIN
        StatBump[statEtherEmptyNoBuffer, missed - lastMissed];
        lastMissed ← missed;
        END;
```

```
    IF pleaseStop THEN EXIT;
    firstInputBuffer ← firstInputBuffer.next;
    SELECT status FROM
      ok =>
        BEGIN
        timeLastRecv ← System.GetClockPulses[];
        IF (new ← GetInputBuffer[]) # NIL THEN
          BEGIN
          new.device ← ethernet;
          this.length ← GetPacketLength[this.iocbChain];
          this.network ← @myNetwork;
          IF doStats THEN
            BEGIN
            etherStats.packetsRecv ← etherStats.packetsRecv + 1;
            StatIncr[statEtherPacketsReceived];
            StatBump[statEtherWordsReceived, this.length];
            END;
          PutOnGlobalInputQueue[this];
          END
        ELSE
          BEGIN
          new ← this; -- Rats, couldn't get a new buffer, recycle this one
          IF doStats THEN
            BEGIN
            etherStats.inputOff ← etherStats.inputOff + 1;
            StatIncr[statEtherEmptyFreeQueue];
            END;
          END;
        END;
      ENDCASE =>
        BEGIN
        new ← this; -- Some kind of error, recycle this buffer
        IF doStats THEN
          SELECT status FROM
            packetTooLong => StatIncr[statEtherReceivedTooLong];
            badAlignmentButOkCrc => StatIncr[statEtherReceivedNot16];
            crc => StatIncr[statEtherReceivedBadCRC];
            crcAndBadAlignment => StatIncr[statEtherReceivedNot16BadCRC];
            overrun =>
              BEGIN
              etherStats.overruns ← etherStats.overruns + 1;
              StatIncr[statEtherReceivedOverrun];
              END;
            ENDCASE => StatIncr[statEtherReceivedBadStatus];
        END;
      -- add new buffer to end of input chain
      QueueInput[
        ether, @new.encapsulation + ethernetEncapsulationOffset,
        new.length - ethernetEncapsulationOffset, new.iocbChain];
      lastInputBuffer.next ← new;
      lastInputBuffer ← new;
      new.next ← NIL;
      ENDLOOP;
  END;

OutInterrupt: ENTRY PROCEDURE =
  BEGIN
  b: Buffer;
  status: Status;
```



```
  Process.SetPriority[5];

  UNTIL pleaseStop DO
    DO
      -- forever until something interesting happens
      IF pleaseStop THEN EXIT;
      -- we compute the values each time around since the value of b can change if
      -- the watcher shoots down the output.
      IF (b ← firstOutputBuffer) # NIL AND (status ← GetStatus[b.iocbChain]) #
        pending THEN EXIT;
      WAIT outWait;
      ENDLOOP;
    IF pleaseStop THEN EXIT; -- so that we do not do something below

    SELECT status FROM
      ok =>
        BEGIN
        IF doStats THEN
          BEGIN
          tries: CARDINAL ← GetRetries[b.iocbChain];
          statEtherSendsCollision1: StatsDefs.StatCounterIndex =
            statEtherSendsCollision1;
          first: CARDINAL = LOOPHOLE[statEtherSendsCollision1];
          etherStats.packetsSent ← etherStats.packetsSent + 1;
          IF tries # 0 THEN StatIncr[LOOPHOLE[first + tries]];
          etherStats.loadTable[tries] ← etherStats.loadTable[tries] + 1;
          END;
        END;
      ENDCASE =>
        BEGIN
        IF doStats THEN
          SELECT status FROM
            tooManyCollisions =>
              BEGIN
              etherStats.loadTable[16] ← etherStats.loadTable[16] + 1;
              StatIncr[statEtherSendsCollisionLoadOverflow];
              END;
            underrun =>
              BEGIN
              etherStats.overruns ← etherStats.overruns + 1;
              StatIncr[statEtherSendOverrun];
              END;
            ENDCASE =>
              BEGIN
              etherStats.badSendSatus ← etherStats.badSendSatus + 1;
              StatIncr[statEtherSendBadStatus];
              END;
        END;

      -- We don't resend things that screwup
      firstOutputBuffer ← firstOutputBuffer.next;
      PutOnGlobalDoneQueue[b];
      ENDLOOP;
  END;

GetElapsedPulses: PROCEDURE [startTime: System.Pulses] RETURNS [System.Pulses] =
  INLINE
  BEGIN
  RETURN[LOOPHOLE[System.GetClockPulses[] - startTime, System.Pulses]];
  END;
```

```
Watcher: PROCEDURE =
  BEGIN
  UNTIL pleaseStop DO
    -- in either case, an interrupt should be pending. Since the interrupt routine is higher priority
**than we are, it should get processed before we can see it. If we get here, an interrupt has proba
**bly been lost. It could have been generated between the time we started decoding the instructi
**on and the time that the data is actually fetched. That is why we look several times. Of course, i
**f it is still not zero when we look again, it could be a new interrupt that has just arrived.
    -- Check for lost input interrupt
    THROUGH [0..25) DO
      IF InputChainOK[] THEN EXIT;
      REPEAT FINISHED => BEGIN WatcherNotify[]; END;
      ENDLOOP;
    -- Check for lost output interrupt
    THROUGH [0..25) DO
      IF OutputChainOK[] THEN EXIT;
      REPEAT FINISHED => BEGIN WatcherNotify[]; END;
      ENDLOOP;
    -- Check for stuck input
    IF GetElapsedPulses[timeLastRecv] > fiveSecondsOfPulses THEN FixupInput[];
    -- Check for stuck output
    IF firstOutputBuffer # NIL AND
      (GetElapsedPulses[timeSendStarted] > fiveHalfSecondsOfPulses) THEN
      ShootDownOutput[];
    WatcherWait[];
    ENDLOOP;
  END;

InputChainOK: ENTRY PROCEDURE RETURNS [BOOLEAN] = INLINE
  BEGIN RETURN[GetStatus[firstInputBuffer.iocbChain] = pending]; END;

OutputChainOK: ENTRY PROCEDURE RETURNS [BOOLEAN] = INLINE
  BEGIN
  RETURN[
    (firstOutputBuffer = NIL) OR
      (GetStatus[firstOutputBuffer.iocbChain] = pending)];
  END;

WatcherWait: ENTRY PROCEDURE = INLINE BEGIN WAIT timer; END;

WatcherNotify: ENTRY PROCEDURE = INLINE
  BEGIN
  IF doStats THEN StatIncr[statEtherLostInterrupts];
  NOTIFY inWait↑;
  NOTIFY outWait↑;
  END;

FixupInput: ENTRY PROCEDURE = INLINE
  BEGIN
  IF doStats THEN StatIncr[statInputIdle];
  SmashCSBs[]; -- this will leave output dangling

  END;

ShootDownOutput: ENTRY PROCEDURE = INLINE
  BEGIN
  -- This happens if the transciever is unplugged
  b: Buffer;
  TurnOff[ether];
```



```
  UNTIL firstOutputBuffer = NIL DO
    b ← firstOutputBuffer;
    firstOutputBuffer ← firstOutputBuffer.next;
    PutOnGlobalDoneQueue[b];
    IF doStats THEN StatIncr[statPacketsStuckInOutput];
    ENDLOOP;
  SmashCSBs[];
  END;

DecapsulateBuffer: PROCEDURE [b: Buffer] RETURNS [BufferType] =
  BEGIN
  SELECT b.encapsulation.ethernetType FROM
    pupEthernetPacket =>
      BEGIN
      IF 2*b.length < b.pupLength + ethernetEncapsulationBytes THEN
        BEGIN
        IF doStats THEN StatIncr[statPupsDiscarded];
        RETURN[rejected];
        END;
      RETURN[pup];
      END;
    oisEthernetPacket =>
      BEGIN
      IF 2*b.length < b.oisPktLength + ethernetEncapsulationBytes THEN
        BEGIN IF doStats THEN StatIncr[statOisDiscarded]; RETURN[rejected]; END;
      RETURN[ois];
      END;
    translationPacket =>
      BEGIN
      IF b.rawWords[0] = translationRequest THEN ReceiveRequest[b]
      ELSE IF b.rawWords[0] = translationResponse THEN ReceiveAck[b];
      RETURN[rejected];
      END;
    ENDCASE => RETURN[rejected];
  END;

EncapsulateRpp: PROCEDURE [RppBuffer, PupHostID] = LOOPHOLE[EncapsulatePup];
EncapsulatePup: PROCEDURE [b: PupBuffer, destination: PupHostID] =
  BEGIN
  b.encapsulation ←
    [ethernet[
      etherSpare1:, etherSpare2:, etherSpare3:, etherSpare4:, etherSpare5:,
      translationWorked: TRUE, etherDest: destination,
      etherSource: myNetwork.hostNumber, ethernetType: pupEthernetPacket]];
  b.length ← (b.pupLength + 1 + ethernetEncapsulationBytes)/2;
  END;

EncapsulateOis: PROCEDURE [
  b: OisBuffer, destination: SpecialSystem.HostNumber] =
  BEGIN
  foundIt: BOOLEAN;
  ethernetAddr: Ethernet1Addr;
  [foundIt, ethernetAddr] ← Translate[destination];
  IF foundIt THEN
    BEGIN
    b.encapsulation ←
      [ethernet[
        etherSpare1:, etherSpare2:, etherSpare3:, etherSpare4:, etherSpare5:,
        translationWorked: TRUE, etherDest: ethernetAddr,
```

```
        etherSource: myNetwork.hostNumber, ethernetType: oisEthernetPacket]];
      b.length ← (b.oisPktLength + 1 + ethernetEncapsulationBytes)/2;
      END
    ELSE
      BEGIN
      b.encapsulation ←
        [ethernet[
          etherSpare1:, etherSpare2:, etherSpare3:, etherSpare4:, etherSpare5:,
          translationWorked: FALSE, etherDest:, etherSource:, ethernetType:]];
      END;
    END;

  ForwardBuffer: PROCEDURE [b: Buffer] RETURNS [PupTypes.PupErrorCode] =
    BEGIN
    IF FALSE THEN -- outputQueue.length>10 THEN
      RETURN[gatewayResourceLimitsPupErrorCode]; -- transceiver unplugged?
    SendBuffer[b];
    RETURN[noErrorPupErrorCode];
    END;

  SendBuffer: ENTRY PROCEDURE [b: Buffer] =
    BEGIN
    IF pleaseStop THEN Glitch[DriverNotActive];
    IF ~b.encapsulation.translationWorked THEN
      BEGIN PutOnGlobalDoneQueue[b]; RETURN; END;
    IF ~hearSelf AND
      (b.encapsulation.etherDest = myNetwork.hostNumber OR
        b.encapsulation.etherDest = PupTypes.allHosts) THEN
      BEGIN -- sending to ourself, copy it over since we can't hear it
      copy: Buffer ← GetInputBuffer[];
      IF copy # NIL THEN
        BEGIN
        copy.device ← ethernet;
        Inline.LongCOPY[
          from: @b.encapsulation + ethernetEncapsulationOffset, nwords: b.length,
          to: @copy.encapsulation + ethernetEncapsulationOffset];
        copy.length ← b.length;
        copy.network ← @myNetwork;
        IF doStats THEN StatIncr[statEtherPacketsLocal];
        IF doStats THEN StatBump[statEtherWordsLocal, b.length];
        PutOnGlobalInputQueue[copy];
        END
      ELSE IF doStats THEN StatIncr[statEtherEmptyFreeQueue];
      END;
    SendBufferInternal[b];
    END;

  SendBufferInternal: INTERNAL PROCEDURE [b: Buffer] =
    BEGIN
    b.device ← ethernet;
    QueueOutput[
      ether, @b.encapsulation + ethernetEncapsulationOffset, b.length,
      b.iocbChain];
    IF doStats THEN StatIncr[statEtherPacketsSent];
    IF doStats THEN StatBump[statEtherWordsSent, b.length];
    IF firstOutputBuffer = NIL THEN firstOutputBuffer ← b
    ELSE lastOutputBuffer.next ← b;
    lastOutputBuffer ← b;
    timeSendStarted ← System.GetClockPulses[];
```



```
    END;

  -- COLD code, only used when turning things on + off

  AdjustLengtoOfD0EthernetInputQueue: PUBLIC PROCEDURE [n: CARDINAL] =
    BEGIN inputQueueLength ← n; END;

  CreateDefaultEthernetOneDrivers: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
    BEGIN
    deviceNumber: CARDINAL ← 0;
    etherDevice: DeviceHandle ← GetNextDevice[nullDeviceHandle];
    IF PilotSwitches.switches.a = down THEN RETURN[FALSE];
    IF etherDevice = nullDeviceHandle THEN RETURN[FALSE];
    WHILE etherDevice # nullDeviceHandle DO
      CreateAnEthernetOneDriver[etherDevice, deviceNumber];
      etherDevice ← GetNextDevice[etherDevice];
      deviceNumber ← deviceNumber + 1;
      ENDLOOP;
    RETURN[TRUE];
    END;

  CreateAnEthernetOneDriver: PROCEDURE [
    etherDevice: DeviceHandle, deviceNumber: CARDINAL] =
    BEGIN
    IF deviceNumber # 0 THEN
      BEGIN
      him: POINTER TO FRAME[EthernetOneDriver] ← NEW EthernetOneDriver;
      him.SetupEthernetOneDriver[etherDevice];
      END
    ELSE SetupEthernetOneDriver[etherDevice];
    END;

  SetupEthernetOneDriver: PROCEDURE [etherDevice: DeviceHandle] =
    BEGIN
    net, host: Environment.Byte;
    ether ← etherDevice;
    [net, host] ← GetEthernet1Address[ether];
    IF net # 0 AND net # myNetwork.netNumber.b AND myNetwork.netNumber #
      nullNetworkNumber THEN Glitch[EthernetNetNumberScrambled];
    IF myNetwork.netNumber = nullNetworkNumber THEN
      myNetwork.netNumber ← [a: 0, b: net];
    pleaseStop ← TRUE;
    myNetwork.buffers ← inputQueueLength;
    AddDeviceToChain[@myNetwork, controlBlockSize];
    IF doStats THEN
      BEGIN
      myNetwork.stats ← etherStats;
      CommUtilDefs.Zero[etherStats, SIZE[EtherStatsInfo]];
      END;
    END;

  ActivateDriver: PROCEDURE =
    BEGIN
    b: Buffer;
    net, host: Environment.Byte;
    IF ~pleaseStop THEN Glitch[DriverAlreadyActive];
    fiveSecondsOfPulses ← System.MicrosecondsToPulses[5000000];
    fiveHalfSecondsOfPulses ← System.MicrosecondsToPulses[2500000];
    [net, host] ← GetEthernet1Address[ether];
```

```
  pleaseStop ← FALSE;
  TurnOff[ether];
  AddCleanup[ether];
  firstInputBuffer ← lastInputBuffer ← NIL;
  firstOutputBuffer ← lastOutputBuffer ← NIL;
  myNetwork.hostNumber ← host;
  THROUGH [0..inputQueueLength) DO
    b ← GetInputBuffer[];
    IF doDebug AND Inline.BITAND[
      Inline.LowHalf[
      LOOPHOLE[@b.encapsulation, LONG CARDINAL] + ethernetEncapsulationOffset],
      3] # 0 THEN Glitch[BufferMustBeQuadWordAligned];
    IF doDebug AND Inline.BITAND[Inline.LowHalf[b.iocbChain], 3] # 0 THEN
      Glitch[IOCBMustBeQuadWordAligned];
    IF doDebug AND Inline.HighHalf[b.iocbChain] # 0 THEN
      Glitch[IOCBMustBeInFirstMDS];
    IF firstInputBuffer = NIL THEN firstInputBuffer ← b;
    IF lastInputBuffer # NIL THEN lastInputBuffer.next ← b;
    lastInputBuffer ← b;
    ENDLOOP;
  [cv: inWait, mask: inInterruptMask] ← ProcessInternal.AllocateNakedCondition[
    ];
  Process.DisableTimeout[inWait];
  [cv: outWait, mask: outInterruptMask] ←
    ProcessInternal.AllocateNakedCondition[];
  Process.DisableTimeout[outWait];
  SmashCSBs[];
  inProcess ← FORK InInterrupt[];
  outProcess ← FORK OutInterrupt[];
  watcherProcess ← FORK Watcher[];
  CreateCache[];
  END;

SmashCSBs: PROCEDURE =
  BEGIN
  b: Buffer;
  TurnOff[ether];
  TurnOn[
    ether, myNetwork.hostNumber, inInterruptMask, outInterruptMask,
    globalStatePtr];
  FOR b ← firstInputBuffer, b.next UNTIL b = NIL DO
    QueueInput[
      ether, @b.encapsulation + ethernetEncapsulationOffset,
      b.length - ethernetEncapsulationOffset, b.iocbChain];
    ENDLOOP;
  timeLastRecv ← System.GetClockPulses[];
  END;

DeleteDriver: PROCEDURE =
  BEGIN
  IF ether # GetNextDevice[nullDeviceHandle] THEN Runtime.SelfDestruct[];
  END;

DeactivateDriver: PROCEDURE =
  BEGIN
  b: Buffer;
  IF pleaseStop THEN Glitch[DriverNotActive];
  pleaseStop ← TRUE;
```



```
  KillInterruptRoutines[];
  JOIN inProcess;
  JOIN outProcess;
  TurnOff[ether];
  ProcessInternal.DeallocateNakedCondition[inWait];
  ProcessInternal.DeallocateNakedCondition[outWait];
  inWait ← outWait ← NIL;
  KillDriverLocked[];
  JOIN watcherProcess;
  RemoveCleanup[ether];
  UNTIL firstInputBuffer = NIL DO
    b ← firstInputBuffer;
    firstInputBuffer ← b.next;
    ReturnFreeBuffer[b];
    ENDLOOP;
  UNTIL firstOutputBuffer = NIL DO
    b ← firstOutputBuffer;
    firstOutputBuffer ← b.next;
    ReturnFreeBuffer[b];
    ENDLOOP;
  myNetwork.netNumber ← nullNetworkNumber;
  -- in case we turn it on after moving to another machine
  DeleteCache[];
  END;

KillInterruptRoutines: ENTRY PROCEDURE = INLINE
  BEGIN NOTIFY inWait↑; NOTIFY outWait↑; END;

KillDriverLocked: ENTRY PROCEDURE = INLINE BEGIN NOTIFY timer; END;

-- Needed by something of Larry's

GetEthernetHostNumber: PUBLIC PROCEDURE RETURNS [CARDINAL] =
  BEGIN
  ether: DeviceHandle ← GetNextDevice[nullDeviceHandle];
  net, host: Environment.Byte;
  [net, host] ← GetEthernet1Address[ether];
  RETURN[host];
  END;


-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- Ugly thing to take care of handling the Ethernet1 (it uses 8 bit addresses).
-- This will go away when OIS Communications stops using Ethernet1's.
-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
-- types

OisAddr: TYPE = SpecialSystem.HostNumber;
Ethernet1Addr: TYPE = PupTypes.PupHostID;
AddressPair: TYPE = MACHINE DEPENDENT RECORD [
  oisAddr: OisAddr, ethernet1Addr: Ethernet1Addr, filler: [0..377B]];

CacheEntry: TYPE = LONG POINTER TO CacheObject;

CacheObject: TYPE = MACHINE DEPENDENT RECORD [
  nextLink: CacheEntry,
  addressPair: AddressPair,
  tries: CARDINAL,
```

```
  timeStamp: System.GreenwichMeanTime,
  status: CacheStatus,
  filler: [0..37777B]];

CacheStatus: TYPE = {new, pending, active, zombie};

-- variables
translationRequest: CARDINAL = 10101B;
translationResponse: CARDINAL = 7070B;
cacheQueueHead: CacheEntry;
broadCastPairEntry: CacheEntry; -- permanent
myAddressPairEntry: CacheEntry; -- permanent
retryLimit: CARDINAL ← 10B;
retryTime: LONG CARDINAL ← 2; -- two seconds
demonActiveTime: Process.Ticks ← Process.SecondsToTicks[1]; -- one second
deactivateTime: LONG CARDINAL ← 3*60; -- three minutes
demonSleepTime: Process.Ticks ← Process.SecondsToTicks[60*5]; -- five minutes
cacheEvent: CONDITION;
demonProcess: PROCESS;


-- interface
CreateCache: ENTRY PROCEDURE = INLINE
  BEGIN
  etherHost: Ethernet1Addr ← Inline.LowHalf[GetEthernetHostNumber[]];
  oisHost: OisAddr ← SpecialSystem.GetProcessorID[];
  aP: AddressPair ← [OISCPTypes.allHostIDs, PupTypes.allHosts, 0];
  Process.SetTimeout[@cacheEvent, demonSleepTime];
  cacheQueueHead ← NIL;
  broadCastPairEntry ← AddAddressPair[aP];
  aP ← [oisHost, etherHost, 0];
  myAddressPairEntry ← AddAddressPair[aP];
  demonProcess ← FORK Demon[];
  END;

DeleteCache: PROCEDURE = INLINE
  BEGIN
  e, nextE: CacheEntry;

  DeleteCacheLocked: ENTRY PROCEDURE = INLINE BEGIN NOTIFY cacheEvent; END;
  -- DeleteCacheLocked

  DeleteCacheLocked[];
  JOIN demonProcess;
  e ← cacheQueueHead;
  cacheQueueHead ← NIL;
  WHILE e # NIL DO nextE ← e.nextLink; Heap.FreeNode[p: e]; e ← nextE; ENDLOOP;
  END;

-- assume protection by lock

depth: CARDINAL;
FindEntry: INTERNAL PROCEDURE [oisAddr: OisAddr] RETURNS [entry: CacheEntry] =
  BEGIN
  IF doStats THEN depth ← 0;
  entry ← cacheQueueHead;
  WHILE entry # NIL DO
    IF oisAddr = entry.addressPair.oisAddr THEN RETURN;
```



```
    entry ← entry.nextLink;
    IF doStats THEN depth ← depth + 1;
    ENDLOOP;
  END;

-- assume protection by lock

AddEntry: INTERNAL PROCEDURE [entry: CacheEntry] =
  BEGIN entry.nextLink ← cacheQueueHead; cacheQueueHead ← entry; END;

-- assume protection by lock

RemoveEntry: INTERNAL PROCEDURE [entry: CacheEntry] =
  BEGIN
  e, pred: CacheEntry;
  IF (pred ← cacheQueueHead) = entry THEN
    BEGIN cacheQueueHead ← cacheQueueHead.nextLink; RETURN; END;
  e ← pred.nextLink;
  WHILE e # NIL DO
    IF e = entry THEN BEGIN pred.nextLink ← entry.nextLink; RETURN; END;
    pred ← e;
    e ← pred.nextLink;
    ENDLOOP;
  ERROR; -- entry not found

  END;


-- locks

Translate: ENTRY PROCEDURE [oisAddr: OisAddr]
  RETURNS [foundIt: BOOLEAN, ethernet1Addr: Ethernet1Addr] =
  BEGIN
  e: CacheEntry;
  IF (e ← FindEntry[oisAddr]) # NIL THEN
    BEGIN
    IF foundIt ← (e.status = active) THEN
      BEGIN
      ethernet1Addr ← e.addressPair.ethernet1Addr;
      e.timeStamp ← System.GetGreenwichMeanTime[];
      END;
    IF e # cacheQueueHead THEN -- put e at the head of the queue
      BEGIN
      IF doStats THEN StatBump[cacheDepth, depth];
      RemoveEntry[e];
      AddEntry[e];
      END;
    END
  ELSE -- entry not found, so add a new one
    BEGIN
    foundIt ← FALSE;
    IF doStats THEN StatIncr[cacheFault];
    e ← Heap.MakeNode[n: SIZE[CacheObject]];
    e.status ← new;
    e.tries ← 0;
    e.timeStamp ← System.GetGreenwichMeanTime[];
    e.addressPair ← [oisAddr: oisAddr, ethernet1Addr:, filler:];
    AddEntry[e];
    NOTIFY cacheEvent;
```

```
    END;
  END;

-- assume protection by lock

AddAddressPair: INTERNAL PROCEDURE [aP: AddressPair] RETURNS [e: CacheEntry] =
  BEGIN
  IF (e ← FindEntry[aP.oisAddr]) = NIL THEN
    BEGIN e ← Heap.MakeNode[n: SIZE[CacheObject]]; AddEntry[e]; END;
  e.addressPair ← aP;
  e.status ← active;
  e.timeStamp ← System.GetGreenwichMeanTime[];
  END;

-- assume protection by lock

DeallocateEntry: INTERNAL PROCEDURE [e: CacheEntry] =
  BEGIN
  -- there are two entries that we do not want to throw out!!
  IF (e = broadCastPairEntry) OR (e = myAddressPairEntry) THEN
    e.timeStamp ← System.GetGreenwichMeanTime[]
  ELSE BEGIN RemoveEntry[e]; Heap.FreeNode[p: e]; END;
  END;


-- locks

Demon: ENTRY PROCEDURE =
  BEGIN
  t: LONG CARDINAL;
  e, nextE: CacheEntry;
  pendingEntries: BOOLEAN;
  Process.SetPriority[5];

  UNTIL pleaseStop DO
    WAIT cacheEvent;
    IF pleaseStop THEN EXIT;
    pendingEntries ← FALSE;
    e ← cacheQueueHead;
    WHILE (e # NIL) DO
      nextE ← e.nextLink;
      t ← System.GetGreenwichMeanTime[] - e.timeStamp;
      SELECT e.status FROM
        active, zombie =>
          BEGIN IF t > deactivateTime THEN DeallocateEntry[e]; END;
        pending =>
          BEGIN
          pendingEntries ← TRUE;
          IF t > retryTime THEN
            BEGIN
            e.tries ← e.tries + 1;
            IF e.tries > retryLimit THEN
              BEGIN
              e.status ← zombie;
              IF doStats THEN StatIncr[unsuccessfulTranslation];
              END
            ELSE
              BEGIN
              IF doStats THEN StatIncr[translationRetries];
              SendRequest[e];
```



```
              e.timeStamp ← System.GetGreenwichMeanTime[];
              END;
            END;
          END;
        new =>
          BEGIN
          pendingEntries ← TRUE;
          SendRequest[e];
          e.status ← pending;
          e.timeStamp ← System.GetGreenwichMeanTime[];
          END;
        ENDCASE => ERROR;
      e ← nextE;
      ENDLOOP; -- end of queue entries loop
    IF pendingEntries THEN Process.SetTimeout[@cacheEvent, demonActiveTime]
    ELSE Process.SetTimeout[@cacheEvent, demonSleepTime];
    ENDLOOP; -- end of infinite loop

  END;


-- assume protection by lock

SendRequest: INTERNAL PROCEDURE [e: CacheEntry] =
  BEGIN
  b: Buffer;
  request: LONG POINTER TO AddressPair;
  IF (b ← MaybeGetFreeBuffer[]) # NIL THEN
    BEGIN
    -- broadcast the trnslation request
    b.encapsulation ←
      [ethernet[
        etherSpare1:, etherSpare2:, etherSpare3:, etherSpare4:, etherSpare5:,
        translationWorked:, etherDest: PupTypes.allHosts,
        etherSource: myNetwork.hostNumber, ethernetType: translationPacket]];
    b.length ← (1 + ethernetEncapsulationBytes)/2 + 2*SIZE[AddressPair] + 1;
    b.rawWords[0] ← translationRequest;
    request ← LOOPHOLE[@b.rawWords[1]];
    request↑ ← e.addressPair;
    -- also send our addresses, so responder does not fault
    request ← request + SIZE[AddressPair];
    request↑ ← myAddressPairEntry.addressPair;
    -- send it
    SendBufferInternal[b];
    END;
  END;

-- locks

ReceiveAck: ENTRY PROCEDURE [b: Buffer] = INLINE
  BEGIN
  IF b.encapsulation.etherDest = myAddressPairEntry.addressPair.ethernet1Addr
    THEN
    BEGIN
    receipt: LONG POINTER TO AddressPair ← LOOPHOLE[@b.rawWords[1]];
    [] ← AddAddressPair[receipt↑];
    END;
  END;
```

```
  -- locks

  ReceiveRequest: ENTRY PROCEDURE [b: Buffer] = INLINE
    BEGIN
    request, requesterAddr: LONG POINTER TO AddressPair;
    request ← LOOPHOLE[@b.rawWords[1]];
    IF request.oisAddr = myAddressPairEntry.addressPair.oisAddr THEN
      BEGIN
      IF doStats THEN StatIncr[requestsForMe];
      request.ethernet1Addr ← myAddressPairEntry.addressPair.ethernet1Addr;
      SendAck[request↑, b.encapsulation.etherSource];
      -- since the requester is probably going to talk to us, add his address before we take a fault
      requesterAddr ← request + SIZE[AddressPair];
      [] ← AddAddressPair[requesterAddr↑];
      END;
    END;

  -- assume protection by lock

  SendAck: INTERNAL PROCEDURE [aP: AddressPair, to: DriverTypes.Byte] = INLINE
    BEGIN
    b: Buffer;
    response: LONG POINTER TO AddressPair;
    IF (b ← MaybeGetFreeBuffer[]) # NIL THEN
      BEGIN
      b.encapsulation ←
        [ethernet[
          etherSpare1:, etherSpare2:, etherSpare3:, etherSpare4:, etherSpare5:,
          translationWorked:, etherDest: to, etherSource: myNetwork.hostNumber,
          ethernetType: translationPacket]];
      b.length ← (1 + ethernetEncapsulationBytes)/2 + SIZE[AddressPair] + 1;
      b.rawWords[0] ← translationResponse;
      response ← LOOPHOLE[@b.rawWords[1]];
      response↑ ← aP;
      -- send it
      SendBufferInternal[b];
      END;
    END;

  -- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  -- End of Ethernet1 uglyness
  -- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


  -- initialization

  Process.SetTimeout[@timer, Process.MsecToTicks[1000]];
  IF doDebug THEN
    BEGIN
    debugPointer: LONG POINTER TO GiantVector ← GetGiantVector[];
    debugPointer.currentInputBuffer ← @firstInputBuffer;
    debugPointer.nextInputBuffer ← @lastInputBuffer;
    debugPointer.currentOutputBuffer ← @firstOutputBuffer;
    END;
  END. -- EthernetOneDriver
```

February 23, 1980 9:10 PM By forrest; Change InlineDefs to Inline, and move EtherStatsInfo Type f
***rom AltoEthernetDefs (RIP) to inside here.*
April 18, 1980 3:00 AM By Murray; Face changes (and other Pilotization).

May 19, 1980 10:20 AM By BLyon; DisableTimeout, SetTimeout, MsecToTicks, SetPriority importe
***d from Process instead of CommUtilDefs.*
May 27, 1980 11:44 AM By BLyon; Added ethernet1 translation uglyness.
August 1, 1980 4:24 PM By BLyon; 8bit->32bit net number Kludge in CreateDefaultEthernetsDriver
***s and use SpecialSystem.nullNetworkNumber instead of 0.*
August 13, 1980 1:13 PM By BLyon; added InputChainOK and OutputChainOK to insure the bodie
***s of the two loops in Watcher are protected by the monitor lock.*
August 25, 1980 1:50 PM By BLyon; multiple ethernets now possible.
August 28, 1980 12:50 PM By McJones; converted to ProcessInternal.AllocateNakedCondition.
September 5, 1980 4:44 PM By HGM; use hearSelf, a switch to ignore boards even if they are ther
***e.*
September 17, 1980 4:41 PM By BLyon; added myNetwork.buffers stuff.

```
-- HalfDuplexImpl.mesa (last edited by: BLyon on: August 20, 1980 4:00 PM) --
DIRECTORY
  BufferDefs: FROM "BufferDefs" USING [Buffer],
  DriverDefs: FROM "DriverDefs" USING [doStats],
  DriverTypes: FROM "DriverTypes" USING [
    Encapsulation, PhonePacketType, phoneEncapsulationOffset,
    phoneEncapsulationBytes],
  HalfDuplex: FROM "HalfDuplex" USING [],
  Process: FROM "Process" USING [SetTimeout, MsecToTicks],
  RS232C: FROM "RS232C" USING [
    ChannelHandle, LineSpeed, DeviceStatus, CompletionHandle, PhysicalRecord,
    ChannelSuspended, Put, TransmitNow, SetParameter, GetStatus, StatusWait],
  SpecialSystem: FROM "SpecialSystem" USING [ProcessorID],
  System: FROM "System" USING [PulsesToMicroseconds, GetClockPulses];

HalfDuplexImpl: MONITOR
  IMPORTS RS232C, System, Process EXPORTS HalfDuplex SHARES BufferDefs =
  BEGIN
  -- types
  ControlMode: TYPE = {unknown, master, slave};
  TurnAroundState: TYPE = {sending, receiving};
  Timer: TYPE = {
    turnAround, -- max time sending in one direction
    noMoreToSend, -- idle sender becomes receiver after this time
    masterResponse};
  -- master sends turn-around if it receives nothing in this time
  StatsRecord: TYPE = RECORD [
    turnArndSent, turnArndRcvd, masterTOs, noSendTOs, turnTOs: CARDINAL];
  -- writeable data
  timerProcess: PROCESS;
  turnAroundState: TurnAroundState;
  controlMode: ControlMode;
  timer: ARRAY Timer OF LONG CARDINAL;
  timeout: ARRAY Timer OF LONG CARDINAL ←
    [turnAround:, noMoreToSend: nothingSentNMTSTimeout,
      masterResponse: initialMasterResponseTimeout];
  stopTimer: BOOLEAN;
  timeoutTimer: CONDITION;
  turnAroundArrived: CONDITION;
  ourTurnToSend: BOOLEAN;
  driverSending: BOOLEAN;
  senderHasMore: BOOLEAN;
  sendFinished: CONDITION;
  channel: RS232C.ChannelHandle;
  clearToSendUp: BOOLEAN;
  ourProcessorID: SpecialSystem.ProcessorID;
  statsRec: StatsRecord;
  -- constants
  timerWakeup: CARDINAL = 250;
  turnAroundPhonePacket: DriverTypes.PhonePacketType = LOOPHOLE[301B];
  moreToSendTurnAroundPhonePacket: DriverTypes.PhonePacketType = LOOPHOLE[302B];
  -- the following are optimized for the current packet stream allocation window size (statically set
**to 3, currently). Our algorithm won't give great performance if window size gets bigger.
  bps1200TurnAroundTimeout: CARDINAL = 13000; -- msecs
  bps2400TurnAroundTimeout: CARDINAL = bps1200TurnAroundTimeout/2; -- msecs
  bps4800TurnAroundTimeout: CARDINAL = bps1200TurnAroundTimeout/4; -- msecs
  bps9600TurnAroundTimeout: CARDINAL = bps1200TurnAroundTimeout/8; -- msecs
  -- noMoreToSend (NMTS) Timeouts
```



```
  nothingSentNMTSTimeout: CARDINAL = 500; -- msecs
  sentSomethingNMTSTimeout: CARDINAL = 250; -- msecs
  remoteAnxiousNMTSTimeout: CARDINAL = 250; -- msecs
  initialMasterResponseTimeout: CARDINAL = 5000; -- msecs
  -- errors and signals
  -- ********Init/termination********
  Initialize: PUBLIC PROCEDURE [
    chHandle: RS232C.ChannelHandle, lineSpeed: RS232C.LineSpeed,
    ourHostNumber: SpecialSystem.ProcessorID] =
    -- initialize timer BOOLEANs and start timer process
    BEGIN
    -- local
    channel ← chHandle; -- save channel handle
    -- the following is to optimize for OISCP windowing, which currently batches 3 packets before r
**equesting an ACK
    timeout[turnAround] ←
      SELECT lineSpeed FROM
        bps2400 => bps2400TurnAroundTimeout,
        bps4800 => bps4800TurnAroundTimeout,
        bps9600 => bps9600TurnAroundTimeout,
        ENDCASE => bps1200TurnAroundTimeout;
    stopTimer ← FALSE;
    clearToSendUp ← FALSE;
    controlMode ← unknown;
    turnAroundState ← receiving;
    ourProcessorID ← ourHostNumber;
    ourTurnToSend ← FALSE;
    driverSending ← FALSE;
    senderHasMore ← FALSE;
    StartTimer[masterResponse];
    Process.SetTimeout[@timeoutTimer, Process.MsecToTicks[timerWakeup]];
    timerProcess ← FORK TimerProcess[];
    IF DriverDefs.doStats THEN statsRec ← [0, 0, 0, 0, 0];
    END;

  Destroy: PUBLIC PROCEDURE =
    -- terminate timer process
    BEGIN
    -- local
    NotifyConditions: ENTRY PROCEDURE =
      BEGIN
      stopTimer ← TRUE;
      ourTurnToSend ← TRUE;
      driverSending ← FALSE;
      NOTIFY timeoutTimer;
      NOTIFY turnAroundArrived;
      NOTIFY sendFinished;
      END;
    NotifyConditions[];
    JOIN timerProcess;
    END;
  -- ********timers********

  TimerProcess: PROCEDURE =
    -- loop checking times, depending on the turnAroundState
    BEGIN
    CheckIfTimedOut: PROCEDURE [timerSelect: Timer] RETURNS [timedOut: BOOLEAN] =
      BEGIN
      timedOut ← (GetClock[] - timer[timerSelect] > timeout[timerSelect]);
```

```
    IF DriverDefs.doStats THEN
      IF timedOut THEN
        SELECT timerSelect FROM
          masterResponse => StatIncr[@statsRec.masterTOs];
          turnAround => StatIncr[@statsRec.turnTOs];
          noMoreToSend => StatIncr[@statsRec.noSendTOs];
          ENDCASE => ERROR;
    END;
  RandomWaitTime: PROCEDURE RETURNS [CARDINAL] = INLINE
    -- gives random number in range [1000..3048)
    BEGIN
    LowExtractor: TYPE = MACHINE DEPENDENT RECORD [
      highBitsLowWord: [0..32), lowTenBits: [0..2048), secondWord: CARDINAL];
    RETURN[(LOOPHOLE[GetClock[], LowExtractor]).lowTenBits + 1000];
    END;
  UNTIL stopTimer DO
    SELECT turnAroundState FROM
      sending =>
        IF CheckIfTimedOut[noMoreToSend] OR CheckIfTimedOut[turnAround] THEN
          BEGIN
          SendLTA[]; -- will happen after any SendFrame in progress
          IF controlMode = unknown THEN
            SetTimer[masterResponse, RandomWaitTime[]];
          IF controlMode # slave THEN StartTimer[masterResponse];
          turnAroundState ← receiving;
          END;
      receiving => -- slave side has no timeout, it just awaits turnaround
        IF controlMode # slave THEN
          BEGIN
          IF CheckIfTimedOut[masterResponse] THEN
            BEGIN -- assume some packet lost
            StartTimer[turnAround];
            SetTimer[noMoreToSend, nothingSentNMTSTimeout];
            -- longer if waiting to send first packet than if we sent one already
            StartTimer[noMoreToSend];
            turnAroundState ← sending;
            senderHasMore ← FALSE;
            IF controlMode # unknown THEN NotifyOurTurnToSend[];
            END;
          END;
      ENDCASE => ERROR;
    WaitTimer[];
    ENDLOOP;
  END;

  WaitTimer: ENTRY PROCEDURE = BEGIN WAIT timeoutTimer; END;

  StartTimer: PROCEDURE [timerSelect: Timer] =
    -- put current time in the timer
    BEGIN timer[timerSelect] ← GetClock[]; END;

  SetTimer: PROCEDURE [timer: Timer, msecs: CARDINAL] =
    -- put current time in the timer
    BEGIN timeout[timer] ← msecs; END;

  GetClock: PROCEDURE RETURNS [msecs: LONG CARDINAL] = INLINE
    -- put current time in the timer
    BEGIN msecs ← System.PulsesToMicroseconds[System.GetClockPulses[]]/1000; END;
```



```
  -- ********Turn-around********

  WaitToSend: PUBLIC PROCEDURE =
    -- wait to be in send mode and for CTS
    BEGIN
    -- locals
    AwaitLineTurnAround: ENTRY PROCEDURE =
      BEGIN
      UNTIL ourTurnToSend DO WAIT turnAroundArrived; ENDLOOP;
      driverSending ← TRUE;
      END;
    AwaitLineTurnAround[];
    AwaitCTS[]; -- set RTS if necessary, wait for CTS

    END;

  SendCompleted: PUBLIC ENTRY PROCEDURE [moreToSend: BOOLEAN] =
    -- tells us we can send LTA now
    BEGIN
    driverSending ← FALSE;
    NOTIFY sendFinished;
    SetTimer[noMoreToSend, sentSomethingNMTSTimeout]; -- reduce timeout
    StartTimer[noMoreToSend];
    senderHasMore ← moreToSend;
    END;

  CheckForTurnAround: PUBLIC PROCEDURE [buffer: BufferDefs.Buffer]
    RETURNS [throwAway: BOOLEAN] =
    -- check packet for line turn-around; determine mode if we need to
    BEGIN OPEN phoneEncap: buffer.encapsulation;
    -- locals
    remoteProcessorID: SpecialSystem.ProcessorID;
    IF controlMode = unknown THEN
      BEGIN
      remoteProcessorID ← LOOPHOLE[phoneEncap.pnSrcID];
      SELECT ourProcessorID.a FROM -- determine mode by comparing processorIDs

        = remoteProcessorID.a =>
          SELECT ourProcessorID.b FROM
            = remoteProcessorID.b =>
              controlMode ←
                (IF ourProcessorID.c > remoteProcessorID.c THEN master
                 ELSE slave);
            > remoteProcessorID.b => controlMode ← master;
            ENDCASE => controlMode ← slave;
        > remoteProcessorID.a => controlMode ← master;
        ENDCASE => controlMode ← slave;
      IF controlMode = master THEN
        SetTimer[masterResponse, initialMasterResponseTimeout];
      END;
    IF phoneEncap.pnType IN
      [turnAroundPhonePacket..moreToSendTurnAroundPhonePacket] THEN
      BEGIN -- a driver turn-around packet (no piggybacking yet)
      IF phoneEncap.pnType = moreToSendTurnAroundPhonePacket THEN
        SetTimer[noMoreToSend, remoteAnxiousNMTSTimeout];
      IF DriverDefs.doStats THEN StatIncr[@statsRec.turnArndRcvd];
      StartTimer[turnAround];
      StartTimer[noMoreToSend];
```

```
    turnAroundState ← sending;
    NotifyOurTurnToSend[]; -- tell anyone waiting to send real packets
    senderHasMore ← FALSE;
    throwAway ← TRUE;
    END
  ELSE
    BEGIN
    throwAway ← FALSE;
    StartTimer[masterResponse]; -- time only from last thing heard

    END;
  END;

NotifyOurTurnToSend: ENTRY PROCEDURE =
  -- tell those waiting to send real packets
  BEGIN ourTurnToSend ← TRUE; NOTIFY turnAroundArrived; END;

AwaitCTS: PROCEDURE =
  -- if modem not ready, make it so (set RTS and await CTS)
  BEGIN
  -- locals
  status: RS232C.DeviceStatus;
  IF ~clearToSendUp THEN
    BEGIN
    ENABLE RS232C.ChannelSuspended => GOTO fatalPlace;
    RS232C.SetParameter[channel, [requestToSend[TRUE]]];
    status ← RS232C.GetStatus[channel];
    UNTIL status.clearToSend DO
      status ← RS232C.StatusWait[channel, status]; ENDLOOP;
    clearToSendUp ← TRUE;
    END;
  EXITS fatalPlace => NULL;
  END;

SendLTA: PROCEDURE =
  -- set flag under monitor and so SendFrame won't send (and not piggyback the turnaround on
**another packet)
  BEGIN
  -- locals
  complHandle: RS232C.CompletionHandle;
  rec: RS232C.PhysicalRecord ←
    [header: [NIL, 0, 0], body:, trailer: [NIL, 0, 0]];
  buffer: DriverTypes.Encapsulation ←
    [phonenet[
      framing0:, framing1:, framing2:, framing3:, framing4:, framing5:,
      recognition: 0,
      pnType:
      (IF senderHasMore THEN moreToSendTurnAroundPhonePacket
       ELSE turnAroundPhonePacket), pnSrcID: LOOPHOLE[ourProcessorID]]];
  ClearOurTurn[];
  AwaitNoSending[];
  AwaitCTS[];
  -- send it, wait for completion, and lower RTS
  rec.body.blockPointer ← @buffer + DriverTypes.phoneEncapsulationOffset;
  -- word boundary
  rec.body.startIndex ← 0;
  rec.body.stopIndexPlusOne ← DriverTypes.phoneEncapsulationBytes;
  BEGIN
  ENABLE RS232C.ChannelSuspended => GOTO fatalPlace;
```



```
  complHandle ← RS232C.Put[channel, @rec];
  [, ] ← RS232C.TransmitNow[channel, complHandle];
  RS232C.SetParameter[channel, [requestToSend[FALSE]]];
  IF DriverDefs.doStats THEN StatIncr[@statsRec.turnArndSent];
  END; -- ENABLE
  clearToSendUp ← FALSE;
  EXITS fatalPlace => NULL;
  END;

ClearOurTurn: ENTRY PROCEDURE = BEGIN ourTurnToSend ← FALSE; END;

AwaitNoSending: ENTRY PROCEDURE =
  BEGIN UNTIL ~driverSending DO WAIT sendFinished; ENDLOOP; END;
-- ********** Statistics **********

StatIncr: PROCEDURE [counter: POINTER TO CARDINAL] =
  -- add one to counter
  BEGIN
  -- locals
  counter↑ ← (counter↑ + 1) MOD (LAST[CARDINAL] - 1);
  END;

StatBump: PROCEDURE [counter: POINTER TO CARDINAL, bumpAmount: CARDINAL] =
  -- add bumpAmount to counter; if bumpAmount < 10000, there will never be overflow
  BEGIN
  -- locals
  counter↑ ← (counter↑ + bumpAmount) MOD (LAST[CARDINAL] - 10000);
  END;
-- MAIN PROGRAM --

END.
```

LOG

Time: July 11, 1980 3:46 PM By: Garlick Action: Created.

*Time: August 11, 1980 3:12 PM By: Garlick Action: Added notification of the turnAround **Arrived and sendFinished conditions in Destroy.*

```
-- NetworkStreamInstance.mesa (last edited by: Garlick on: September 26, 1980 2:19 PM)
-- Function: The implementation module for an instance of the Network Stream front end.

DIRECTORY
  ByteBlt USING [ByteBlt],
  Environment USING [Byte],
  NetworkStreamInternal USING [ControlObject],
  OISCPDefs USING [GetOisPacketTextLength, SetOisPacketTextLength],
  PacketStream USING [
    bytesPerLevel2SppHeader, Destroy, Get, GetSendSppBuffer, Handle, Put,
    ReturnGetSppBuffer, ReturnGetSppDataBuffer, ReturnSendSppBuffer, SppBuffer,
    WaitForAttention],
  Runtime USING [SelfDestruct],
  Stream USING [
    Byte, Word, Handle, Block, CompletionCode, defaultInputOptions, InputOptions,
    LongBlock, Object, ShortBlock, SSTChange, SubSequenceType, TimeOut];

NetworkStreamInstance: PROGRAM [psH: PacketStream.Handle] RETURNS [Stream.Handle]
  IMPORTS ByteBlt, OISCPDefs, Stream, PacketStream, Runtime =
  BEGIN

  -- the vector of procedures for operating on, and controlling the network stream
  controlObject: NetworkStreamInternal.ControlObject ←
    [
      -- the vector of procedures as per the standard Pilot Stream interface
      streamObject: Stream.Object[
      options: Stream.defaultInputOptions, getByte: GetByte, putByte: PutByte,
      getWord: GetWord, putWord: PutWord, get: GetBlock, put: PutBlock,
      setSST: SetSST, sendAttention: SendAttention, waitAttention: WaitAttention,
      delete: Delete],
      -- handle for the packet stream
      psH: psH];

  LeftAndRight: TYPE = MACHINE DEPENDENT RECORD [left, right: Environment.Byte];

  -- A client will typicaly have three processes accessing this module. The first
  -- receives data, the second waits for attentions, and the third transmits data, sends
  -- attentions and changes the subsequence type. As a consequence it is not necessary
  -- for this module to be a monitor, since there is no interaction between the three
  -- processes in this module. The interaction occurs in the PktStreamInstance module,
  -- which is a monitor. Multiple client processes must not perform data transfer in one
  -- direction; the result is unpredicatble. Care should be taken when deleting the stream
  -- and therefore this module.

  -- input
  inputBuffer: PacketStream.SppBuffer ← NIL;
  inputFinger: CARDINAL;
  inputSST: Stream.SubSequenceType ← 0;
  -- output
  outputBuffer: PacketStream.SppBuffer ← NIL;
  outputFinger: CARDINAL;
  outputSST: Stream.SubSequenceType ← 0;
  outputSSTSent: BOOLEAN ← TRUE;
  outputBufferSize: CARDINAL ← 0;

  -- Hot Procedures

  GetByte: PROCEDURE [sH: Stream.Handle] RETURNS [byte: Stream.Byte] =
```



```
    BEGIN
    IF inputBuffer # NIL AND inputFinger + 2 < GetSppDataLength[inputBuffer] THEN
      BEGIN -- " + 2" lets GetBlock give back the buffer if we take the last byte
      byte ← inputBuffer.sppBytes[inputFinger];
      inputFinger ← inputFinger + 1;
      RETURN;
      END
    ELSE
      BEGIN
      array: PACKED ARRAY [0..1] OF Stream.Byte;
      [] ← sH.get[sH, [@array, 0, 1], [FALSE, FALSE, FALSE, TRUE, TRUE]];
      RETURN[array[0]];
      END;
    END;

  GetWord: PROCEDURE [sH: Stream.Handle] RETURNS [word: Stream.Word] =
    BEGIN OPEN w: LOOPHOLE[word, LeftAndRight];
    w.left ← GetByte[sH];
    w.right ← GetByte[sH];
    END;

  -- This procedure fills a client's block with data from an incoming packet

  GetBlock: PROCEDURE [
    sH: Stream.Handle, block: Stream.Block, options: Stream.InputOptions]
    RETURNS [
      bytesTransferred: CARDINAL, why: Stream.CompletionCode,
      sst: Stream.SubSequenceType] =
    -- block has been passed by value, so we are upadting our copy, not the clients
    BEGIN
    input: Stream.Block;
    moved: CARDINAL;
    bytesTransferred ← 0;
    why ← normal;
    sst ← inputSST;
    WHILE block.startIndex < block.stopIndexPlusOne DO
      UNTIL inputBuffer # NIL DO
        inputFinger ← 0;
        inputBuffer ← PacketStream.Get[psH];
        IF inputBuffer = NIL THEN SIGNAL Stream.TimeOut[block.startIndex]
        ELSE
          BEGIN
          sst ← inputBuffer.subtype;
          IF inputSST # sst THEN
            BEGIN
            inputSST ← sst;
            IF options.signalSSTChange THEN
              SIGNAL Stream.SSTChange[inputSST, block.startIndex]
            ELSE BEGIN why ← sstChange; RETURN; END;
            END;
          END;
      ENDLOOP;
      input ←
        [blockPointer: @inputBuffer.sppBytes, startIndex: inputFinger,
          stopIndexPlusOne: GetSppDataLength[inputBuffer]];
      moved ← ByteBlt.ByteBlt[block, input];
      bytesTransferred ← bytesTransferred + moved;
      block.startIndex ← block.startIndex + moved;
      inputFinger ← inputFinger + moved;
```

```
    -- if the packet buffer is empty return it
    IF inputFinger = input.stopIndexPlusOne THEN
      BEGIN
      PacketStream.ReturnGetSppDataBuffer[psH, inputBuffer];
      inputBuffer ← NIL;
      END;
    -- if there is no packet buffer and the block is still empty maybe signal
    IF inputBuffer = NIL AND block.startIndex < block.stopIndexPlusOne AND
      options.signalLongBlock THEN
      BEGIN SIGNAL Stream.LongBlock[block.startIndex]; END;
    -- if there is no packet buffer then maybe the client wants to know
    IF inputBuffer = NIL AND options.terminateOnEndPhysicalRecord THEN
      BEGIN why ← endRecord; EXIT; END;
    ENDLOOP;
  -- if there is data in the packet buffer then the block was short
  IF inputBuffer # NIL AND options.signalShortBlock THEN
    BEGIN ERROR Stream.ShortBlock; END;
  END; -- GetBlock

-- The strategy on the transmission side, is to allocate a buffer only when there is
-- data to be copied into the buffer, or if an empty packet must be transmitted.
-- State information is kept around for things like whether a new SST has been sent
-- to the other end or not, incase it is changed without sending any intervening data.


-- This procedure sends a client's block of data in one or more packets.
-- Since SendNow isn't a procedure all by itself, we try to know when the client
-- did a SendNow, so that we can ask the other end for an ack. If we have no
-- buffer, we send a data packet with zero data.

PutBlock: PROCEDURE [
  sH: Stream.Handle, block: Stream.Block, endPhysicalRecord: BOOLEAN] =
  -- block has been passed by value, so we are updating our copy, not the clients
  BEGIN
  sendNowBlock: Stream.Block = [NIL, 0, 0];
  output: Stream.Block;
  moved: CARDINAL;
  IF (block = sendNowBlock AND endPhysicalRecord) THEN
    BEGIN -- this must be a SendNow operation, or something like it sigh...
    SendNow[];
    RETURN;
    END;
  -- see whether this is a no-op or not.
  IF (block.stopIndexPlusOne - block.startIndex) = 0 AND outputBuffer = NIL AND
    outputSSTSent THEN RETURN;
  IF outputBufferSize = 0 THEN outputBufferSize ← psH.getSenderSizeLimit[];
  WHILE block.startIndex < block.stopIndexPlusOne DO
    IF outputBuffer = NIL THEN
      BEGIN
      outputBuffer ← PacketStream.GetSendSppBuffer[psH];
      outputBuffer.sendAck ← FALSE;
      outputBuffer.attention ← FALSE;
      outputFinger ← 0;
      END;
    output ←
      [blockPointer: @outputBuffer.sppBytes, startIndex: outputFinger,
        stopIndexPlusOne: outputBufferSize];
    moved ← ByteBlt.ByteBlt[output, block];
```



```
    block.startIndex ← block.startIndex + moved;
    outputFinger ← outputFinger + moved;
    IF outputFinger = outputBufferSize THEN FlushOutputBuffer[];
    ENDLOOP;
  IF endPhysicalRecord THEN FlushOutputBuffer[];
  END; -- PutBlock

PutByte: PROCEDURE [sH: Stream.Handle, byte: Stream.Byte] =
  BEGIN
  IF outputBuffer # NIL AND outputFinger + 2 < outputBufferSize THEN
    BEGIN -- " + 2" lets PutBlock flush the buffer if we fill the last byte
    outputBuffer.sppBytes[outputFinger] ← byte;
    outputFinger ← outputFinger + 1;
    RETURN;
    END
  ELSE
    BEGIN
    array: PACKED ARRAY [0..1] OF Stream.Byte ← [byte, ];
    PutBlock[sH, [@array, 0, 1], FALSE];
    END;
  END;

PutWord: PROCEDURE [sH: Stream.Handle, word: Stream.Word] =
  BEGIN OPEN w: LOOPHOLE[word, LeftAndRight];
  sH.putByte[sH, w.left];
  sH.putByte[sH, w.right];
  END;

SendNow: PROCEDURE =
  BEGIN
  IF outputBuffer = NIL THEN
    BEGIN
    outputBuffer ← PacketStream.GetSendSppBuffer[psH];
    outputBuffer.attention ← FALSE;
    outputFinger ← 0;
    END;
  outputBuffer.sendAck ← TRUE;
  FlushOutputBuffer[];
  END; -- SendNow

-- This procedure sets the SST to the specified value and has some side effects.
-- We assume that the SST is initially = 0, and the first change causes no empty packet
-- to be sent

SetSST: PROCEDURE [sH: Stream.Handle, sst: Stream.SubSequenceType] =
  BEGIN
  IF sst # outputSST THEN
    BEGIN
    FlushOutputBuffer[]; -- flush the last buffer if there was one
    IF NOT outputSSTSent THEN
      -- there was no buffer to flush for the old SST so send an empty packet
      BEGIN
      outputBuffer ← PacketStream.GetSendSppBuffer[psH];
      outputBuffer.sendAck ← FALSE;
      outputBuffer.attention ← FALSE;
      outputFinger ← 0;
      FlushOutputBuffer[];
      END;
    -- remember the new SST
    outputSST ← sst;
```

```
    outputSSTSent ← FALSE
    END;
  END; -- SetSSt

-- This procedure sends one byte of data in a packet with the attention bit set.

SendAttention: PROCEDURE [sH: Stream.Handle, byte: Stream.Byte] =
  BEGIN
  FlushOutputBuffer[]; -- flush the last buffer if there was one
  outputBuffer ← PacketStream.GetSendSppBuffer[psH];
  outputBuffer.sppBytes[0] ← byte;
  outputFinger ← 1;
  outputBuffer.sendAck ← FALSE;
  outputBuffer.attention ← TRUE;
  FlushOutputBuffer[];
  END; -- SendAttention

-- This procedure waits indefinately until an attention arrives, or an ERROR is raised.

WaitAttention: PROCEDURE [sH: Stream.Handle] RETURNS [byte: Stream.Byte] =
  BEGIN
  b: PacketStream.SppBuffer;
  DO
    b ← PacketStream.WaitForAttention[psH];
    -- Get the first data byte, if there is one, otherwise discard this attention packet.
    -- Discarding the attention packet upsets the timeout mechanism, but then this is
    -- a situation that is not supposed to happen, and so we pay the price.
    IF GetSppDataLength[b] # 0 THEN
      BEGIN
      byte ← b.sppBytes[0];
      PacketStream.ReturnGetSppBuffer[psH, b];
      EXIT;
      END
    ELSE PacketStream.ReturnGetSppBuffer[psH, b];
    ENDLOOP;
  END; -- WaitAttention

-- This procedure flushes (i.e. sends out) the outputBuffer if there is one.

FlushOutputBuffer: PROCEDURE =
  BEGIN
  b: PacketStream.SppBuffer;
  -- don't leave outputBuffer dangling in case of Stream deletion
  IF outputBuffer = NIL THEN RETURN;
  b ← outputBuffer;
  outputBuffer ← NIL;
  SetSppDataLength[b, outputFinger];
  b.subtype ← outputSST;
  outputSSTSent ← TRUE;
  PacketStream.Put[psH, b];
  END; -- FlushOutputBuffer

-- This procedure sets the length of the sequenced packet given the length of data..

SetSppDataLength: PROCEDURE [b: PacketStream.SppBuffer, length: CARDINAL] =
  INLINE
  BEGIN
  OISCPDefs.SetOisPacketTextLength[
    b, length + PacketStream.bytesPerLevel2SppHeader];
```



```
  END; -- SetSppDataLength

-- This procedure returns the amount of data in the sequenced packet.

GetSppDataLength: PROCEDURE [b: PacketStream.SppBuffer] RETURNS [CARDINAL] =
  INLINE
  BEGIN
  RETURN[
    OISCPDefs.GetOisPacketTextLength[b] - PacketStream.bytesPerLevel2SppHeader];
  END; -- GetSppDataLength


-- Cool Procedures

-- This procedure is instrumental in deleting this transducer.

Delete: PUBLIC PROCEDURE [sH: Stream.Handle] =
  BEGIN
  IF inputBuffer # NIL THEN
    PacketStream.ReturnGetSppDataBuffer[psH, inputBuffer];
  IF outputBuffer # NIL THEN
    PacketStream.ReturnSendSppBuffer[psH, outputBuffer];
  PacketStream.Destroy[psH];
  Runtime.SelfDestruct[];
  END; -- Delete


-- initialization (Cool)

RETURN[@controlObject.streamObject];
END. -- of NetworkStreamInstance module
```

LOG

Time: May 26, 1978 11:22 AM By: Dalal Action: created file.
Time: November 9, 1978 9:07 AM By: Dalal Action: modified GetBlock and PutBlock.
Time: March 13, 1979 6:03 PM By: Dalal Action: modified SendAttention and WaitAttention.
Time: August 31, 1979 12:39 PM By: Dalal Action: made two INLINEs.
Time: January 31, 1980 4:33 AM By: Forrest Action: Added Mandatory fields to Stream.Object, us
***ing Stream Defaults.*
Time: July 8, 1980 5:07 PM By: BLyon Action: removed default procs and replaced with GetByte,
***GetWord, PutByte, PutWord.*
Time: August 11, 1980 4:57 PM By: BLyon Action: Changed ByteBltDefs to ByteBlt
Time: September 26, 1980 2:19 PM By: Garlick Action: In GetBlock, initialized sst to inputSST so i
***t always gets returned with proper value.*

```
-- NetworkStreamMgr.mesa (last edited by: BLyon on: September 29, 1980 3:51 PM)
-- Function: The implementation module for the manager of Pilot Network Streams.

DIRECTORY
  BufferDefs USING [BufferPool],
  CommunicationInternal USING [],
  NetworkStream USING [
    ConnectionID, ConnectionFailed, ConnectionSuspended, unknownConnID, WaitTime,
    CloseStatus, closeSST, closeReplySST],
  NetworkStreamInstance USING [],
  NetworkStreamInternal USING [CloseState, ControlHandle],
  OISCPDefs USING [OisAddress, SppBuffer, ReturnReceiveSppBufferToPool],
  PacketStream USING [
    ConnectionAlreadyThere, FindAddresses, Handle, Make, OisConnectionID,
    SetWaitTime],
  Router USING [XmitStatus],
  Socket USING [
    Abort, AssignNetworkAddress, ChannelAborted, ChannelError, ChannelHandle,
    Delete, SetWaitTime, TimeOut],
  SocketInternal USING [
    CreateInternal, GetBufferPool, GetPacket, SocketHandle,
    SocketHandleToChannelHandle],
  SpecialSystem USING [NetworkAddress, nullNetworkAddress],
  Stream USING [
    Byte, Handle, PutByte, GetByte, SendNow, SetSST, SSTChange, TimeOut],
  System USING [];

NetworkStreamMgr: PROGRAM
  IMPORTS
    NetworkStream, netStrmInst: NetworkStreamInstance, OISCPDefs, PacketStream,
    Socket, SocketInternal, Stream
  EXPORTS CommunicationInternal, NetworkStream, System
  SHARES BufferDefs, NetworkStreamInstance =
  BEGIN OPEN NetworkStream;

  -- EXPORTED TYPE(S) and READONLY Variables
  ListenerHandle: PUBLIC TYPE = SocketInternal.SocketHandle;
  NetworkAddress: PUBLIC TYPE = SpecialSystem.NetworkAddress;
  uniqueNetworkAddr: PUBLIC NetworkAddress ← SpecialSystem.nullNetworkAddress;

  IllegalAddress: PUBLIC ERROR = CODE;

  -- Cool Procedures

  -- This procedure creates a network stream to the specified remote address.
  Create: PUBLIC PROCEDURE [remote: NetworkAddress, timeout: WaitTime]
    RETURNS [Stream.Handle] =
    BEGIN
    RETURN[
      CreateTransducer[
        uniqueNetworkAddr, remote, NetworkStream.unknownConnID,
        NetworkStream.unknownConnID, TRUE, timeout]];
    END; -- Create

  AssignNetworkAddress: PUBLIC PROCEDURE RETURNS [NetworkAddress] =
    BEGIN RETURN[Socket.AssignNetworkAddress[]]; END; -- AssignNetworkAddress

  -- This procedure returns the local and remote addresses of the Network Stream.
```



```
  FindAddresses: PUBLIC PROCEDURE [sH: Stream.Handle]
    RETURNS [local, remote: NetworkAddress] =
    BEGIN
    [local, remote] ← PacketStream.FindAddresses[
      LOOPHOLE[sH, NetworkStreamInternal.ControlHandle].psH];
    END; -- FindAddresses

  -- This procedure sets the wait time for the Network Stream.

  SetWaitTime: PUBLIC PROCEDURE [sH: Stream.Handle, time: WaitTime] =
    BEGIN
    PacketStream.SetWaitTime[
      LOOPHOLE[sH, NetworkStreamInternal.ControlHandle].psH, time];
    END; -- SetWaitTime

  -- This procedure creates a listener at the specified local Network Address.
  -- The generation of IllegalAddress will become more sophisticated.
  -- We enqueue all the buffers onto the socket channel.

  CreateListener: PUBLIC PROCEDURE [addr: NetworkAddress]
    RETURNS [lH: ListenerHandle] =
    BEGIN
    listenerSendBuffers: CARDINAL = 0;
    listenerReceiveBuffers: CARDINAL = 2;
    lH ← SocketInternal.CreateInternal[
      addr, normal, listenerSendBuffers, listenerReceiveBuffers !
      Socket.ChannelError => GOTO bad];
    EXITS bad => RETURN WITH ERROR IllegalAddress;
    END; -- CreateListener

  -- This procedure deletes a listener.

  DeleteListener: PUBLIC PROCEDURE [listenerH: ListenerHandle] =
    BEGIN
    -- We assume that there are no processes doing a Listen.
    -- The Listen code is such that it always does a Socket.Get leaving the buffer with
    -- the socket channel.
    Socket.Abort[SocketInternal.SocketHandleToChannelHandle[listenerH]];
    Socket.Delete[SocketInternal.SocketHandleToChannelHandle[listenerH]];
    END; -- DeleteListener


  -- This procedure creates a sequenced packet transducer with all its parameters.

  CreateTransducer: PUBLIC PROCEDURE [
    local, remote: NetworkAddress,
    localConnID, remoteConnID: NetworkStream.ConnectionID,
    activelyEstablish: BOOLEAN, timeout: NetworkStream.WaitTime]
    RETURNS [sH: Stream.Handle] =
    BEGIN
    psH: PacketStream.Handle;
    newNetStrmInst: POINTER TO FRAME[NetworkStreamInstance];
    psH ← PacketStream.Make[
      local, remote, LOOPHOLE[localConnID, PacketStream.OisConnectionID],
      LOOPHOLE[remoteConnID, PacketStream.OisConnectionID], activelyEstablish,
      timeout];
    newNetStrmInst ← NEW netStrmInst;
    sH ← START newNetStrmInst[psH];
    END; -- CreateTransducer
```

```
-- These procedures define an optional close protocol using the exchange of reserved
-- subsequence types on the client's behalf.

-- This procedure closes communication over a network stream at the client's
-- level of protocol. The semantics are I want to close the stream, therefore I do
-- not want to transmit any more data and only want to know that the remote
-- end is aware of my actions. Any input data arriving on the stream may be discarded.
-- The status is good, noReply if the other end just did not respond, and incomplete if it
-- was a simultaneous close and the third data packet did not make it and the other
-- end is gone.

Close: PUBLIC PROCEDURE [sH: Stream.Handle]
  RETURNS [status: NetworkStream.CloseStatus] =
  BEGIN
  state: NetworkStreamInternal.CloseState ← sendClose;
  dataByte: Stream.Byte ← 0;
  status ← good;
  Stream.SetSST[sH, NetworkStream.closeSST];
  Stream.PutByte[sH, dataByte];
  Stream.SendNow[sH];
  state ← waitCloseReply;
  UNTIL state = closed OR state = sendCloseReply DO
    dataByte ← Stream.GetByte[
      sH !
      Stream.SSTChange =>
        BEGIN
        IF sst = NetworkStream.closeReplySST THEN
          BEGIN
          -- tricky we are diddling the stream from within a catch phrase
          Stream.SetSST[sH, NetworkStream.closeReplySST];
          Stream.PutByte[sH, dataByte];
          Stream.SendNow[sH];
          state ← closed;
          END
        ELSE IF sst = NetworkStream.closeSST THEN state ← sendCloseReply;
        -- simultaneous close
        RESUME
        ;
        END;
      Stream.TimeOut => BEGIN status ← noReply; state ← closed; EXIT; END;
      NetworkStream.ConnectionSuspended =>
        BEGIN status ← noReply; state ← closed; EXIT; END];
    ENDLOOP;
  IF state = sendCloseReply THEN status ← CloseReply[sH];
  END; -- Close

-- This procedure conforms to the close protocol and is invoked by the client of
-- a network stream when it receives a subsequence change to closeSST. The
-- semantics of this call are that the client knows that the other end wants to
-- close the communication and it will not transmit any more data and will reply
-- with a closeReplySST. The status can be good, or incomplete; the latter implying
-- that an answer to the closeReplySST did not make it back and the other end is
-- gone, or that the closeReply was never acked.

CloseReply: PUBLIC PROCEDURE [sH: Stream.Handle]
  RETURNS [status: NetworkStream.CloseStatus] =
  BEGIN
  state: NetworkStreamInternal.CloseState ← sendCloseReply;
  dataByte: Stream.Byte ← 0;
```



```
  status ← good;
  Stream.SetSST[sH, NetworkStream.closeReplySST];
  Stream.PutByte[sH, dataByte];
  Stream.SendNow[sH];
  state ← waitCloseReplyReply;
  UNTIL state = closed DO
    dataByte ← Stream.GetByte[
      sH !
      Stream.SSTChange =>
        BEGIN
        IF sst = NetworkStream.closeReplySST THEN state ← closed;
        RESUME
        ;
        END;
      Stream.TimeOut => BEGIN status ← incomplete; state ← closed; EXIT; END;
      NetworkStream.ConnectionSuspended =>
        BEGIN status ← noReply; state ← closed; EXIT; END];
    ENDLOOP;
  END; -- CloseReply

-- Hot Procedures

-- This procedure listens on the specified socket channel for a sequenced packet, and
-- creates a network stream from another unique local socket to the remote end.
-- This procedure checks to make sure that the arriving packet is not a duplicate.

Listen: PUBLIC PROCEDURE [
  listenerH: ListenerHandle,
  listenTimeout, streamTimeout: NetworkStream.WaitTime]
  RETURNS [Stream.Handle] =
  BEGIN OPEN SocketInternal;
  pool: BufferDefs.BufferPool ← GetBufferPool[listenerH];
  b: OISCPDefs.SppBuffer;
  sH: Stream.Handle;
  Socket.SetWaitTime[SocketHandleToChannelHandle[listenerH], listenTimeout];
  DO
    -- until the packet is a sequenced packet and is not a duplicate, or timeout, or aborted socket
    sH ← NIL;
    b ← LOOPHOLE[GetPacket[
      listenerH ! Socket.TimeOut => EXIT; Socket.ChannelAborted => EXIT],
      OISCPDefs.SppBuffer];
    -- b.status can be one of the errors or goodCompletion
    IF LOOPHOLE[b.status, Router.XmitStatus] = goodCompletion THEN
      BEGIN
      IF b.packetType = sequencedPacket THEN
        BEGIN
        -- examine this packet
        IF b.destinationConnectionID = unknownConnID AND b.sourceConnectionID #
          unknownConnID AND b.sequenceNumber = 0 AND b.systemPacket THEN
          BEGIN
          -- good packet, but must see if this is an old duplicate
          IF NOT PacketStream.ConnectionAlreadyThere[
            b.source, b.sourceConnectionID] THEN
            BEGIN
            sH ← CreateTransducer[
              uniqueNetworkAddr, b.source, NetworkStream.unknownConnID,
              LOOPHOLE[b.sourceConnectionID, NetworkStream.ConnectionID], TRUE,
              streamTimeout !
              ConnectionFailed => BEGIN sH ← NIL; CONTINUE; END];
```

```
            OISCPDefs.ReturnReceiveSppBufferToPool[pool, b];
            EXIT;
            END;
          END;
        END;
      END;
    OISCPDefs.ReturnReceiveSppBufferToPool[pool, b];
    ENDLOOP;
  RETURN[sH];
  END; -- Listen


  -- initialization (Cold)

  END. -- NetworkStreamMgr module
```

LOG

Time: May 9, 1978 1:12 PM By: Dalal Action: created file.
Time: August 31, 1979 1:14 PM By: Dalal Action: modified Listener routines.
Time: January 26, 1980 2:51 PM By: Dalal Action: moved stuff to PacketStreamMgr.
Time: April 15, 1980 10:22 AM By: BLyon Action: uses new Level1 routines.
Time: July 15, 1980 11:11 AM By: BLyon Action: Use exported types.
Time: September 29, 1980 3:51 PM By: BLyon Action: Listner catches Socket.ChannelAborted.

```
-- PacketStreamInstance.mesa (last edited by: Garlick on: October 12, 1980 5:30 PM)
-- Function: The implementation module for an instance of a SPP packet stream.

DIRECTORY
  BufferDefs USING [BufferPool, QueueLength, QueueCleanup, QueueInitialize],
  ByteBlt USING [ByteBlt],
  DriverDefs USING [doDebug, doStats, Glitch],
  Environment USING [Block],
  Inline USING [LowHalf],
  OISCPDefs USING [
    DequeueSpp, EnqueueSpp, OisBuffer, QueueObject,
    MaybeGetFreeSendSppBufferFromPool, MaybeGetFreeReceiveSppBufferFromPool,
    ReturnReceiveSppBufferToPool, ReturnReceiveOisBufferToPool,
    ReturnSendSppBufferToPool],
  PacketStream: FROM "PacketStream",
  Process USING [MsecToTicks, SetTimeout, Ticks, Yield],
  Router USING [XmitStatus],
  Runtime USING [SelfDestruct],
  Socket USING [Abort, ChannelAborted, Delete, GetStatus, TimeOut],
  SocketInternal USING [
    CreateInternal, GetBufferPool, GetPacket, PutPacket, SocketHandle,
    SocketHandleToChannelHandle],
  SpecialSystem USING [NetworkAddress],
  StatsDefs USING [StatBump, StatIncr],
  System USING [GetClockPulses, MicrosecondsToPulses, PulsesToMicroseconds];

-- Time units have been converted from MilliSeconds to Pulses; However, all interface
-- still are in units of MilliSeconds (like the timeout passed to the MONITOR).
-- GetClock now returns Pulses. Variable names ending in 'Time' will be in milliseconds
-- (hopefully only initial values), while other variables will be in Pulses (some will end in
-- 'Pulses').

PacketStreamInstance: MONITOR [
  localAddr, remoteAddr: PacketStream.OisAddress,
  localConnectionID, remoteConnectionID: PacketStream.OisConnectionID,
  establish: BOOLEAN, timeout: PacketStream.WaitTime]
  RETURNS [
    PacketStream.Handle, PacketStream.OisAddress, PacketStream.OisConnectionID]
  IMPORTS
    BufferDefs, ByteBlt, DriverDefs, Inline, Process, Runtime, Socket,
    SocketInternal, StatsDefs, System, PacketStream, OISCPDefs
  EXPORTS System
  SHARES BufferDefs =
  BEGIN OPEN DriverDefs, OISCPDefs, Router, PacketStream, StatsDefs;

  -- EXPORTED TYPE(S)
  NetworkAddress: PUBLIC TYPE = SpecialSystem.NetworkAddress;

  sendBufs: CARDINAL = 6;
  receiveBufs: CARDINAL = 6;

  -- client interface
  ps: Object ←
    [NIL, TakeFromUser, GetForUser, WaitForAttention, SetWaitTime, FindAddresses,
      GetSenderSizeLimit, ReturnGetSppDataBuffer]; -- pool (NIL) initted later

  -- connection control parameters
  -- connection state
```



```
  state: PacketStream.State;
  whySuspended: PacketStream.SuspendReason;
  whyFailed: PacketStream.FailureReason;
  stateBeforeSuspension: PacketStream.State;
  connectionIsEstablished: CONDITION;
  -- socket interface data
  socketCH: SocketInternal.SocketHandle;
  pool: BufferDefs.BufferPool;
  -- this is the buffer pool from which we get buffers
  -- sequencing, duplicate suppression and flow control information
  nextInputSeq, maxInputSeq: CARDINAL;
  unackedOutputSeq, nextOutputSeq, maxOutputSeq: CARDINAL;
  newAllocation: CONDITION;
  sendAnAck: BOOLEAN;
  -- connection control
  maxOisPktLength: CARDINAL;
  -- input attention control
  newInputAttn: CONDITION;
  -- table to suppress attentions that the client has seen, but we haven't acked.
  -- we permit only a small number of pending attns, the rest are thrown away!
  attnSeqTable: ARRAY (0..maxPendingAttns] OF CARDINAL;
  attnCount: CARDINAL;
  sentBuffer: SppBuffer; -- hold the head of the sentQueue
  -- input/output pointers and queues
  inOrderQueue, inAttnQueue, outOfOrderQueue, tempQueue, sentQueue:
    OISCPDefs.QueueObject;
  inOrderQueueNotEmpty: CONDITION;
  -- variables and parameters for this packet stream
  ackRequestPulse: LONG CARDINAL; -- the time an ack was requested, a Pulse
  lastPacketReceivedPulse: LONG CARDINAL;
  -- time any packet was rcv'd for this connection, a Pulse
  lastProbePulse: LONG CARDINAL; -- retransmitter's last probe sent time, a Pulse
  probeCounter: CARDINAL; -- retransmitter's allocation probe counter
  probeRetransmitPulses: LONG CARDINAL; -- in Pulses
  dataPacketRetransmitPulses: LONG CARDINAL; -- in Pulses
  waitPulses: LONG CARDINAL;
  -- maximum time (in Pulses) any blocking procedure should block
  delayCount: CARDINAL; -- number of data packets used in delay calculation
  delaySum: LONG CARDINAL;
  -- cumulative time (in Pulses) that delayCount packets spent on retransmit queue
  lastDelayCalculationPulse: LONG CARDINAL;
  -- a Pulses, last time we updated retransmit time
  seqNumWhenDelayCalculated: CARDINAL;
  -- used to see if data flowed in retrans stuff.
  -- process handles and specific parameters for controlling them
  pleaseStop: BOOLEAN;
  letClientRun: CONDITION;
  retransmitterFork, receiverFork: PROCESS;
  retransmitterWakeupTimer: CONDITION;
  -- temporary stats
  normalRetransUpdates: CARDINAL; -- **temp
  doubleRetransUpdates: CARDINAL; -- **temp

  -- constants for performance
  -- 12 packet buffers at the socket.
  -- 4 are for sending, 4 for receiving.
  -- 1 for sending a system packet and 1 for receiving a system packet
  -- 1 for receiving an attention packet
```

```
-- 1 for extra receive packet
defaultSendAllocation: CARDINAL = 4;
defaultRcvAllocation: CARDINAL = 4;
duplicateWindow: INTEGER = 100;
maxPendingAttns: CARDINAL = 5;
-- some day these will become variables whose value will be determined heuristically
-- using some kind of adaptive algorithm
retransmissionsBeforeGiveUp: CARDINAL = 10;
retransmissionsBeforeAskForAck: CARDINAL = 2;
probesBeforeGiveUp: CARDINAL = 4;
emptyInOrderQueue: INTEGER = 0;
minSendAllocationBeforeRequestingAck: INTEGER = 0;
minRcvAllocation: INTEGER = 0;
initialDataPacketRetransmitTime: CARDINAL = 2000; -- msecs
maxDataPacketRetransmitTime: LONG CARDINAL = 20000; -- msecs
maxDataPacketRetransmitPulses: LONG CARDINAL ← MilliSecondsToPulses[
  maxDataPacketRetransmitTime];
initialRetransmitterTime: CARDINAL = 250; -- msecs
probeMultiplier: CARDINAL = 4; -- probe time is this multiple of retrans timeout
delayCalculationTime: LONG CARDINAL = 10000;
-- 10 secs (keep < 1 min to assure that delaySum doesn't overflow)
delayCalculationPulses: LONG CARDINAL ← MilliSecondsToPulses[
  delayCalculationTime];
inactiveConnectionTime: LONG CARDINAL = 5000; -- msecs
inactiveConnectionPulses: LONG CARDINAL ← MilliSecondsToPulses[
  inactiveConnectionTime];
fortyMsecsOfPulses: LONG CARDINAL ← MilliSecondsToPulses[40];
fiveHundredMsecsOfPulses: LONG CARDINAL ← MilliSecondsToPulses[500];
-- the following four constants are weights used in UpdateRetransmitTime
wOldAbove: CARDINAL = 1;
wOldBelow: CARDINAL = 2;
wNewAbove: CARDINAL = 50;
wNewBelow: CARDINAL = 2;


-- We are using an adaptive scheme for deciding when packets must be retransmitted.
-- We assume that dataPacketRetransmitPulses must elapse since the last time
-- ackRequestPulse was updated before retransmitting everything on the sentQueue.
-- ackRequestPulse is updated whenever a packet with the sendAck bit set is
-- transmitted or retransmitted, or when a packet is received that updates the
-- unackedOutputSeq field.  We transmit
-- a probe whenever ackRequestPulse hasn't been updated for 5 sec.  The connection
-- can become suspended if we send 4 probes without a response (probes being
-- generated because of allocation requirements or the 5 sec threshold), or if we
-- retransmit a packet too many times.  This stuff is very tricky and need careful
-- thought for the next Pilot.


-- various Glitches generated by this module
StreamNotEstablished: ERROR = CODE;
StreamTerminating: ERROR = CODE;

-- Hot Procedures

-- Gives the clock time in machine dependent Pulses
GetClock: PROCEDURE RETURNS [LONG CARDINAL] = INLINE
  BEGIN
  -- System.GetClockPulses can be subtracted if interval between calls is less than the
```



```
  -- wraparound (which is processor dependent and > 1 day on the D* machines)
  RETURN[LOOPHOLE[System.GetClockPulses[], LONG CARDINAL]];
  END;

-- This procedure gives a Sequenced Packet Protocol packet to be transmitted over
-- the specified packet stream.  This packet resides in a buffer, and some of the fields
-- of the packet must be filled in by the caller of this procedure.  These fields are
-- oisPktLength, the attention flag, and subtype.  This buffer will be returned to the
-- free queue once the packet has been acknowledged.  SIGNALs generated by this
-- procedure leave the clean up responsibility for the buffer to the caller.

TakeFromUser: PROCEDURE [b: SppBuffer] =
  BEGIN

  TakeFromUserLocked: ENTRY PROCEDURE = INLINE
    BEGIN
    ENABLE UNWIND => NULL;
    DO
      SELECT state FROM
        unestablished, activeEstablish, waitEstablish =>
          IF doDebug THEN Glitch[StreamNotEstablished];
        established, open =>
          IF LOOPHOLE[nextOutputSeq - maxOutputSeq, INTEGER] <= 0 THEN
            BEGIN
            b.systemPacket ← FALSE;
            -- we should ask for an acknowledgement to be returned, if on sending this
            -- packet there is space only for minAllocationBeforeRequestingAck, or this is
            -- an attention:  We should do this only if, our client hasn't already set the bit.
            IF NOT b.sendAck THEN
              b.sendAck ← b.attention OR LOOPHOLE[maxOutputSeq - nextOutputSeq,
                INTEGER] = minSendAllocationBeforeRequestingAck;
            b.unusedType ← 0;
            PrepareSequencedPacket[b];
            END
          ELSE
            BEGIN
            WAIT newAllocation;
            LOOP; -- we start from the beginning in case the state changed

            END;
        suspended => RETURN WITH ERROR ConnectionSuspended[whySuspended];
        terminating => IF doDebug THEN Glitch[StreamTerminating];
        ENDCASE;
      EXIT;
      ENDLOOP;
    END; -- TakeFromUserLocked

  TakeFromUserLocked[]; -- either returns or generates a SIGNAL.
  IF doStats THEN StatIncr[statDataPacketsSent];
  IF doStats THEN
    StatBump[statDataBytesSent, b.oisPktLength - bytesPerSequencedPktHeader];
  PutPacketOnSocketChannel[b];
  END; -- TakeFromUser

-- This procedure gets the next sequenced, duplicate-suppressed packet from the
-- packet stream.  This procedure hangs till a packet is available.  The client is
-- expected to return the buffer to the free queue.  If the wait times expires then
-- NIL will be returned, and so the client MUST test for this case.
```

```
GetForUser: PROCEDURE RETURNS [b: SppBuffer] =
  BEGIN
  returnAnAck: BOOLEAN ← FALSE;

  GetForUserLocked: ENTRY PROCEDURE = INLINE
    BEGIN
    ENABLE UNWIND => NULL;
    startPulse, now: LONG CARDINAL;
    startPulse ← GetClock[];
    DO
      SELECT state FROM
        unestablished, activeEstablish, waitEstablish =>
          IF doDebug THEN Glitch[StreamNotEstablished];
        established, open =>
          BEGIN
          IF (b ← DequeueSpp[@inOrderQueue]) # NIL THEN
            BEGIN
            IF sendAnAck AND BufferDefs.QueueLength[@inOrderQueue] =
              emptyInOrderQueue THEN
              BEGIN
              sendAnAck ← FALSE; -- set the global switch to false
              returnAnAck ← TRUE; -- we must send back the ack

              END;
            IF doStats THEN
              BEGIN
              StatIncr[statDataPacketsReceived];
              StatBump[
                statDataBytesReceived,
                b.oisPktLength - bytesPerSequencedPktHeader];
              END;
            END
          ELSE
            BEGIN
            now ← GetClock[];
            IF (now - startPulse) >= waitPulses THEN EXIT;
            WAIT inOrderQueueNotEmpty;
            LOOP; -- we start from the beginning in case the state changed

            END;
          END;
        suspended => RETURN WITH ERROR ConnectionSuspended[whySuspended];
        terminating => IF doDebug THEN Glitch[StreamTerminating];
        ENDCASE;
      EXIT;
      ENDLOOP;
    END; -- GetForUserLocked

  GetForUserLocked[];
  IF returnAnAck THEN SendSystemPacket[FALSE];
  END; -- GetForUser

-- This procedure waits until a packet with the attention bit arrives, and returns
-- an advance copy to the client, who must return it to the buffer pool. The client will
-- not get duplicates, though it may see the advance copies of the attention
-- packet after it has seen the real copy if it is sluggish.

WaitForAttention: ENTRY PROCEDURE RETURNS [b: SppBuffer] =
  BEGIN
  ENABLE UNWIND => NULL;
```



```
  startPulse, now: LONG CARDINAL;
  startPulse ← GetClock[];
  DO
    SELECT state FROM
      unestablished, activeEstablish, waitEstablish =>
        IF doDebug THEN Glitch[StreamNotEstablished];
      established, open =>
        BEGIN
        IF (b ← DequeueSpp[@inAttnQueue]) # NIL THEN EXIT
        ELSE
          BEGIN
          now ← GetClock[];
          IF (now - startPulse) >= waitPulses THEN
            RETURN WITH ERROR AttentionTimeout;
          WAIT newInputAttn;
          LOOP; -- we start from the beginning in case the state changed

          END;
        END;
      suspended => RETURN WITH ERROR ConnectionSuspended[whySuspended];
      terminating => RETURN WITH ERROR AttentionTimeout;
      -- until Process.Abort implemented

      ENDCASE;
    EXIT;
    ENDLOOP;
  END; -- WaitForAttention

-- This procedure returns a processed spp data buffer to its packet stream socket's pool.

ReturnGetSppDataBuffer: ENTRY PROCEDURE [b: SppBuffer] =
  BEGIN
  maxInputSeq ← maxInputSeq + 1;
  OISCPDefs.ReturnReceiveSppBufferToPool[pool, b];
  END;

-- This procedure causes the transmission of a system packet, i.e. one that does not
-- consume any sequence number. The subtype of this packet is irrelavent and so is
-- made zero. The procedure takes an argument indicating whether the send
-- acknowledgement flag should be set or not in the outgoing packet.

SendSystemPacket: PROCEDURE [sendAck: BOOLEAN] =
  BEGIN
  b: SppBuffer;

  SendSystemPacketLocked: ENTRY PROCEDURE = INLINE
    BEGIN PrepareSequencedPacket[b]; END; -- SendSystemPacketLocked

    IF (b ← OISCPDefs.MaybeGetFreeSendSppBufferFromPool[pool]) # NIL THEN
    BEGIN
    -- b.requeueProcedure should ony be set and inspected by level1 and network drivers.
    b.oisPktLength ← bytesPerSequencedPktHeader;
    b.systemPacket ← TRUE;
    b.sendAck ← sendAck;
    b.attention ← FALSE;
    b.unusedType ← 0;
    b.subtype ← 0;
    SendSystemPacketLocked[];
    PutPacketOnSocketChannel[b];
```

```
    END;
  END; -- SendSystemPacket

-- This procedure takes a buffer and prepares it for transmission by filling in the
-- appropriate fields in the buffer and updating the packet stream state variables.

PrepareSequencedPacket: INTERNAL PROCEDURE [b: SppBuffer] = INLINE
  BEGIN
  -- b.requeueProcedure should ony be set and inspected by level1 and network drivers
  -- b.length filled by router
  -- b.network filled by router
  -- b.iocbChain filled by network driver
  -- b.userPtr not used currently
  -- b.userData not used currently
  -- b.userDataLength not used currently
  -- b.status used by the socket and router code
  -- b.time used to determine round-trip delay.  Perhaps someday individual retransmit timeout.
**Filled when put on retransmit queue
  b.tries ← 0;
  -- rest used by queue package, drivers and dispatcher
  -- encapsulation filled by router
  -- b.checksum filled by router
  -- b.oisPktLength filled by caller of this routine
  -- b.transportControl filled by router
  b.packetType ← sequencedPacket;
  b.destination ← remoteAddr;
  -- b.source filled by socket interface
  -- b.systemPacket filled by caller of this routine
  -- b.sendAck filled by acknowledgement strategy by caller
  -- b.attention filled by caller of this routine
  -- b.unusedType filled by caller of this routine
  -- b.subtype filled by caller of this routine
  b.sourceConnectionID ← localConnectionID;
  b.destinationConnectionID ← remoteConnectionID;
  b.sequenceNumber ← nextOutputSeq;
  b.acknowledgeNumber ← nextInputSeq;
  b.allocationNumber ← maxInputSeq;
  IF NOT b.systemPacket THEN nextOutputSeq ← nextOutputSeq + 1
  ELSE
    BEGIN
    IF doStats THEN StatIncr[statAcksSent];
    IF doStats THEN StatIncr[statSystemPacketsSent];
    END;
  IF b.sendAck AND doStats THEN StatIncr[statAckRequestsSent];
  IF b.attention AND doStats THEN StatIncr[statAttentionsSent];
  END; -- PrepareSequencedPacket

-- This procedure puts a packet out on the socket channel, where it waits for the
-- transmission to be complete.  On completion the procedure enqueues the packet onto
-- the sentQueue for possible retransmission only if the packet is not a system packet.
-- The packet stream monitor should not be locked when we call this.

PutPacketOnSocketChannel: PROCEDURE [b: SppBuffer] =
  BEGIN

  EnqueueTransmittedPacketAppropriately: ENTRY PROCEDURE = INLINE
    BEGIN

    FailConnection: INTERNAL PROCEDURE [why: FailureReason] =
```



```
      BEGIN
      SuspendStream[noRouteToDestination];
      whyFailed ← why; -- inform the creator if there is one

      END; -- FailConnection

    -- remember time for retransmitter
    IF b.sendAck THEN ackRequestPulse ← GetClock[];
    -- check for failures (this may be the first packet, so look for connection failures)
    SELECT LOOPHOLE[b.status, XmitStatus] FROM
      pending, goodCompletion, aborted, invalidDestAddr => NULL;
      -- ok or not expected

      noRouteToNetwork, hardwareProblem => FailConnection[noRouteToDestination];
      noTranslationForDestination =>
        FailConnection[noTranslationForDestination];
      noAnswerOrBusy => FailConnection[noAnswerOrBusy];
      circuitInUse => FailConnection[circuitInUse];
      circuitNotReady => FailConnection[circuitNotReady];
      noDialingHardware => FailConnection[noDialingHardware];
      dialerHardwareProblem => FailConnection[noRouteToDestination];
      ENDCASE => ERROR;
    -- enqueue onto sentQueue if not a system packet else return to free queue
    IF b.systemPacket THEN OISCPDefs.ReturnSendSppBufferToPool[pool, b]
    ELSE
      BEGIN
      EnqueueSpp[@sentQueue, b];
      IF b.tries = 0 AND b.sendAck THEN b.time ← GetClock[];
      -- record time if ackReq packet and first time put there (used for delay calculation)

      END;
    END; -- EnqueueTransmittedPacketAppropriately

  SocketInternal.PutPacket[
    socketCH, LOOPHOLE[b, OisBuffer] ! Socket.ChannelAborted => CONTINUE];
  -- we are about to be deleted
  EnqueueTransmittedPacketAppropriately[];
  END; -- PutPacketOnSocketChannel

-- This procedure changes the state of the packet stream to be suspended.

SuspendStream: INTERNAL PROCEDURE [why: SuspendReason] =
  BEGIN
  stateBeforeSuspension ← state;
  state ← suspended;
  whySuspended ← why;
  END; -- SuspendStream

-- This procedure changes the state of the packet stream to be suspended.

SuspendStreamLocked: ENTRY PROCEDURE [why: SuspendReason] =
  BEGIN SuspendStream[why]; END; -- SuspendStreamLocked

-- This process retransmits packets that have not been acknowledged in a reasonable
-- time, and in addition generates probes etc. to test for the connection's livenesss.
-- The Retransmitter, Receiver and the transmission process all three try to keep the
-- sentBuffer and sentQueue ordered by sequence number.
```

```
Retransmitter: PROCEDURE =
  BEGIN
  now: LONG CARDINAL;
  retransBuffer: SppBuffer;

  -- some procedures whose scope is just the Retransmitter

  TimeToSendAProbe: ENTRY PROCEDURE RETURNS [goAhead: BOOLEAN] = INLINE
    BEGIN
    goAhead ← FALSE;
    IF state = established OR state = open THEN
      BEGIN
      IF
        ((LOOPHOLE[nextOutputSeq - maxOutputSeq, INTEGER] > 0 AND
          (now - lastProbePulse) > probeRetransmitPulses)) OR
        (((now - lastPacketReceivedPulse) >= inactiveConnectionPulses AND
          (now - lastProbePulse) >= inactiveConnectionPulses)) THEN
        BEGIN
        -- probe for allocation, or just an ack from other end, or for activity.
        lastProbePulse ← now;
        IF (probeCounter ← probeCounter + 1) > probesBeforeGiveUp THEN
          SuspendStream[transmissionTimeout]
        ELSE BEGIN goAhead ← TRUE; StatIncr[statProbesSent]; END;
        END;
      END;
    END; -- TimeToSendAProbe

  RetransmissionCount: ENTRY PROCEDURE RETURNS [CARDINAL] = INLINE
    BEGIN
    IF state = established OR state = open THEN
      BEGIN
      IF (now - ackRequestPulse) >= dataPacketRetransmitPulses THEN
        RETURN[sentQueue.length + (IF sentBuffer = NIL THEN 0 ELSE 1)]
        -- we must retransmit all packets currently on the sentQueue and sentBuffer.
        -- a count is kept because on retransmission we put packets back on
        -- sentQueue and otherwise we will be in an infinite loop.

      ELSE RETURN[0]; -- not yet time to retransmit

      END
    ELSE RETURN[0]; -- any other state requires no retransmission.

    END; -- RetransmissionCount

  GetFromRetransmissionQueue: ENTRY PROCEDURE RETURNS [b: SppBuffer] = INLINE
    BEGIN
    IF sentBuffer = NIL THEN b ← DequeueSpp[@sentQueue]
    ELSE BEGIN b ← sentBuffer; sentBuffer ← NIL; END;
    IF b = NIL THEN RETURN;
    SELECT b.tries FROM
      >= retransmissionsBeforeGiveUp =>
        BEGIN -- give up trying to send anything on this packet stream.
        SuspendStream[transmissionTimeout];
        sentBuffer ← b; -- put back at head of sentQueue
        b ← NIL;
        END;
      = retransmissionsBeforeAskForAck => b.sendAck ← TRUE;
      -- need to set ack request

      ENDCASE;
```



```
    END; -- GetFromRetransmissionQueue

  WaitForAWhile: ENTRY PROCEDURE = INLINE
    BEGIN WAIT retransmitterWakeupTimer; END; -- WaitForAWhile

  UpdateRetransmitTime: PROCEDURE = INLINE
    BEGIN
    avgDelay: LONG CARDINAL;
    --weightNewAbove: CARDINAL;
    IF ((now - lastDelayCalculationPulse) >= delayCalculationPulses) THEN
      BEGIN -- time to do retransmission timeout determination
      IF delayCount > 0 THEN
        BEGIN
        -- we have had some packets with lossless delay (no retransmissions)
        avgDelay ← delaySum/delayCount;
        normalRetransUpdates ← normalRetransUpdates + 1; -- **temp
        -- new retransmit timeout is a function of the old retransmit timeout and the
        --newly measured delay. If the delay is less than about 500 msecs, the standard
        --deviation of the measured delay is large compared to transmission time;
        --otherwise transmission time dominates. So for fast mediums, we multiply the
        --delay by a large factor. For slow mediums, we go with the measured delay.
        -- The following lines need some work; they have been commmented out and a
        -- temporay estimation added.
        --weightNewAbove ← IF avgDelay > fiveHundredMsecsOfPulses THEN 1 ELSE wNewAbov
**e;
        --dataPacketRetransmitPulses ←
        --(dataPacketRetransmitPulses / wOldBelow)* wOldAbove +
        --(avgDelay / wNewBelow)* weightNewAbove;
        dataPacketRetransmitPulses ←
          (dataPacketRetransmitPulses + wNewAbove*avgDelay)/2;
        delaySum ← 0;
        delayCount ← 0;
        END
      ELSE
        BEGIN
        -- if data flowed, all had retranmissions and we should increase timeout
        IF unackedOutputSeq > seqNumWhenDelayCalculated THEN
          BEGIN
          doubleRetransUpdates ← doubleRetransUpdates + 1; -- **temp
          dataPacketRetransmitPulses ← dataPacketRetransmitPulses*2;
          END;
        END;
      seqNumWhenDelayCalculated ← unackedOutputSeq;
      dataPacketRetransmitPulses ← MIN[
        dataPacketRetransmitPulses, maxDataPacketRetransmitPulses];
      probeRetransmitPulses ←
        probeMultiplier*dataPacketRetransmitPulses + fortyMsecsOfPulses;
      Process.SetTimeout[
        @retransmitterWakeupTimer, PulsesToTicks[dataPacketRetransmitPulses/4]];
      -- set new retransmitter wakeup

      END;
    END; -- UpdateRetransmitTime

  -- main body of the procedure
  UNTIL pleaseStop DO
    -- checks the value of pleaseStop, race condition OK
    now ← GetClock[];
    IF TimeToSendAProbe[] THEN SendSystemPacket[TRUE];
```

```
    THROUGH (0..RetransmissionCount[]] DO
      -- this is an upper bound
      -- state may change for any reason, but we keep sending them out, unless we
      -- ourselves changed the state to be suspended! The Receiver may take packets
      -- off the sentQueue in parallel in which case we will get a NIL and therefore exit.
      IF (retransBuffer ← GetFromRetransmissionQueue[]) = NIL THEN EXIT;
      IF LOOPHOLE[retransBuffer.sequenceNumber - unackedOutputSeq, INTEGER] < 0
        THEN OISCPDefs.ReturnSendSppBufferToPool[pool, retransBuffer]
        -- packet has been ACKed

      ELSE
        BEGIN
        -- its time to send this packet out again
        retransBuffer.tries ← retransBuffer.tries + 1;
        PutPacketOnSocketChannel[retransBuffer]; -- requeues buffer on sentQueue
        retransBuffer ← NIL;
        IF doStats THEN StatIncr[statDataPacketsRetransmitted];
        END;
      ENDLOOP;
    WaitForAWhile[];
    UpdateRetransmitTime[];
    ENDLOOP;
  END; -- Retransmitter

PulsesToTicks: PROCEDURE [pulses: LONG CARDINAL] RETURNS [Process.Ticks] =
  INLINE
  BEGIN
  lastCard: LONG CARDINAL = LAST[CARDINAL];
  msecs: LONG CARDINAL ← System.PulsesToMicroseconds[LOOPHOLE[pulses]]/1000 + 1;
  RETURN[
    Process.MsecToTicks[
      IF msecs > lastCard THEN LAST[CARDINAL] ELSE Inline.LowHalf[msecs]]];
  END;

-- This process waits at the socket channel for a packet to arrive.
-- This packet is in a buffer that belongs to this socket and should be returned to the
-- pool associated with the socket channel.

Receiver: PROCEDURE =
  BEGIN
  b: OisBuffer;
  UNTIL pleaseStop DO
    -- race condition not harmful as we account for it
    -- now wait for something to arrive.
    b ← SocketInternal.GetPacket[
      socketCH ! Socket.TimeOut => RETRY; Socket.ChannelAborted => EXIT];
    -- pleaseStop got set by someone else
    SELECT LOOPHOLE[b.status, XmitStatus] FROM
      goodCompletion =>
        SELECT b.packetType FROM
          sequencedPacket => GotSequencedPacket[LOOPHOLE[b, SppBuffer]];
          error => GotErrorPacket[b];
          ENDCASE =>
            BEGIN
            -- discard, don't want any other protocol type
            OISCPDefs.ReturnReceiveOisBufferToPool[pool, b];
            IF doStats THEN StatIncr[statPacketsRejectedBadType];
            END;
      ENDCASE => OISCPDefs.ReturnReceiveOisBufferToPool[pool, b];
```



```
    ENDLOOP;
  END; -- Receiver

-- This procedure examines the sequenced packet that has just arrived.

GotSequencedPacket: PROCEDURE [b: SppBuffer] =
  BEGIN
  returnAnAck: BOOLEAN ← FALSE;
  -- we do not return an ack unless we are required to
  giveBackToSocket: BOOLEAN ← TRUE; -- unless we put on some other queue
  nowPulse: LONG CARDINAL; -- set if we get a packet for our connection

  GotSequencedPacketLocked: ENTRY PROCEDURE = INLINE
    BEGIN
    SELECT state FROM
      unestablished, suspended, terminating => NULL; -- not interested.

      activeEstablish, waitEstablish =>
        [returnAnAck, giveBackToSocket] ← EstablishThisConnection[b];
      established, open =>
        BEGIN
        -- check that packet is from right remote address and connection ID
        IF b.source.host = remoteAddr.host AND b.source.socket =
          remoteAddr.socket AND b.sourceConnectionID = remoteConnectionID AND
          b.destinationConnectionID = localConnectionID THEN
          BEGIN -- the packet is for this connection
          probeCounter ← 0; -- other end is alive
          lastPacketReceivedPulse ← nowPulse ← GetClock[];
          -- for inactivity et al
          SELECT LOOPHOLE[b.sequenceNumber - nextInputSeq, INTEGER] FROM
            -- just the packet we wanted, or packet is early

            IN [0..duplicateWindow] => RightOrEarlyPacket[];
            -- old duplicate packet

            IN [-duplicateWindow..0) => DuplicatePacket[];
            -- very old duplicate packet

            ENDCASE => IF doStats THEN StatIncr[statDataPacketsReceivedVeryLate];
          END
        ELSE
          -- packet is really not for this connection (or other end not established)
          BEGIN
          IF b.source.host # remoteAddr.host OR b.source.socket #
            remoteAddr.socket THEN
            IF doStats THEN StatIncr[statPacketsRejectedBadSource];
          IF b.sourceConnectionID # remoteConnectionID OR
            b.destinationConnectionID # localConnectionID THEN
            BEGIN
            IF doStats THEN StatIncr[statPacketsRejectedBadID];
            returnAnAck ← TRUE; -- probably our connection response got lost

            END;
          END;
        END;
    ENDCASE;
    -- this is delicate and is done to avoid getting stuck in a small window
    IF returnAnAck AND BufferDefs.QueueLength[@inOrderQueue] # emptyInOrderQueue
      THEN
```

```
    BEGIN
    returnAnAck ← FALSE; -- we don't have to send back an ack
    sendAnAck ← TRUE; -- make another process send back an ack

    END;
  -- This wait lets the user process run and suck up all the packets on the
  -- inOrderInputQueue so that we send back a full allocate.
  IF LOOPHOLE[maxInputSeq - nextInputSeq, INTEGER] < minRcvAllocation THEN
    WAIT letClientRun;
  END; -- GotSequencedPacketLocked

-- This procedure processes both the next expected packet as well as early packets.
-- Both kinds of packets are processed similarly as far as acks and allocation,
-- and attentions are concerned.
RightOrEarlyPacket: INTERNAL PROCEDURE = INLINE
  BEGIN
  IF b.sendAck THEN
    BEGIN
    returnAnAck ← TRUE;
    IF doStats THEN StatIncr[statAckRequestsReceived];
    END;
  -- update allocation and ack fields of packetstream.
  IF LOOPHOLE[b.allocationNumber - maxOutputSeq, INTEGER] > 0 THEN
    BEGIN maxOutputSeq ← b.allocationNumber; NOTIFY newAllocation; END;
  IF LOOPHOLE[b.acknowledgeNumber - unackedOutputSeq, INTEGER] > 0 THEN
    BEGIN
    unackedOutputSeq ← b.acknowledgeNumber;
    ackRequestPulse ← nowPulse; -- if we get an ack we move the ack req. time

    END;
  -- now remove acked packets, if any, from sentQueue. We assume that
  -- the sentQueue is kept ordered by increasing sequence number.
  IF sentBuffer = NIL THEN sentBuffer ← DequeueSpp[@sentQueue];
  UNTIL sentBuffer = NIL OR
    LOOPHOLE[sentBuffer.sequenceNumber - unackedOutputSeq, INTEGER] >= 0 DO
    IF sentBuffer.sendAck AND sentBuffer.tries = 0 THEN
      BEGIN -- constitutes a measure of lossless delay (no retransmissions)
      delaySum ← delaySum + (nowPulse - sentBuffer.time);
      -- add time spent on queue to delay stats
      delayCount ← delayCount + 1;
      END;
    OISCPDefs.ReturnSendSppBufferToPool[pool, sentBuffer];
    sentBuffer ← DequeueSpp[@sentQueue];
    ENDLOOP;
  -- so far we have only updated state information, now to dispense with the packet
  IF b.systemPacket THEN
    BEGIN
    IF doStats THEN
      BEGIN
      StatIncr[statSystemPacketsReceived];
      StatIncr[statAcksReceived];
      END;
    END
  ELSE
    -- this is not a system packet process it intelligently
    BEGIN
    -- if the attention bit is set then we try to process the attention, and only
    -- if we were able to do so do we put the packet on the inOrderQueue.
    IF ~b.attention OR (b.attention AND AttentionPacketProcessed[b]) THEN
      BEGIN
```



```
      -- only now must we decide if it is in order or out of order
      IF LOOPHOLE[b.sequenceNumber - nextInputSeq, INTEGER] = 0 THEN
        -- just the packet we wanted
        BEGIN
        state ← open;
        DO
          nextInputSeq ← nextInputSeq + 1;
          IF attnCount # 0 THEN UpdateAttnSeqTable[];
          EnqueueSpp[@inOrderQueue, b];
          giveBackToSocket ← FALSE;
          NOTIFY inOrderQueueNotEmpty;
          -- examine the OutOfOrderQueue to see if anything can be taken off
          -- currently there is never anything on the OutOfOrderQueue.
          DO
            IF BufferDefs.QueueLength[@outOfOrderQueue] = 0 THEN
              BEGIN b ← NIL; EXIT; END;
            b ← DequeueSpp[@outOfOrderQueue];
            SELECT LOOPHOLE[b.sequenceNumber - nextInputSeq, INTEGER] FROM
              0 => EXIT;
              -- this is the next in order packet, pick it up in outer loop

              IN (0..duplicateWindow] => EnqueueSpp[@tempQueue, b];
              -- still out of order

              ENDCASE => OISCPDefs.ReturnReceiveSppBufferToPool[pool, b];
            -- old duplicate, throw away

            ENDLOOP;
          -- put buffers that are on tempQueue back onto outOfOrderQueue
          UNTIL (tempQueue.first) = NIL DO
            EnqueueSpp[@outOfOrderQueue, DequeueSpp[@tempQueue]]; ENDLOOP;
          IF b = NIL THEN EXIT;
          -- else, loop to process packet from outOfOrderQueue that is now next

          ENDLOOP;
        END
      ELSE IF doStats THEN StatIncr[statDataPacketsReceivedEarly];
      -- early packet

      END;
    END;
  END; -- RightOrEarlyPacket

DuplicatePacket: INTERNAL PROCEDURE = INLINE
  BEGIN
  IF doStats THEN StatIncr[statDataPacketsReceivedAgain];
  -- send acks only if the other end asked
  IF b.sendAck THEN returnAnAck ← TRUE;
  END; -- DuplicatePacket

-- main body of this sprocedure
IF b.oisPktLength >= bytesPerSequencedPktHeader THEN
  BEGIN
  GotSequencedPacketLocked[];
  IF returnAnAck THEN SendSystemPacket[FALSE]; -- we may have to send the ack

  END
ELSE IF doStats THEN StatIncr[statEmptyFunnys];
IF giveBackToSocket THEN OISCPDefs.ReturnReceiveSppBufferToPool[pool, b];
```

```
  END; -- GotSequencedPacket

-- This procedure attempts to processes the attention packet, and returns whether it
-- was successful or not. Success implies that an entry for the attn packet already
-- exists in the attnSeqTable, or has just been made. This packet is now a candidate
-- for the inOrderQueue or the outOfOrderQueue. If an entry can not be made because
-- of space restrictions we pretend we never ever saw this packet and let it be
-- retransmitted from the source. The attnSeqTable is reasonablly big, and so this
-- decision should not cause adverse effects. This procedure is only called if the packet
-- was within the accept window.

AttentionPacketProcessed: INTERNAL PROCEDURE [b: SppBuffer] RETURNS [BOOLEAN] =
  BEGIN
  i: CARDINAL ← 1;
  alreadyThere: BOOLEAN ← FALSE;
  advanceB: SppBuffer;
  to, from: Environment.Block;
  nBytes: CARDINAL;
  IF attnCount = maxPendingAttns THEN RETURN[FALSE]; -- no space
  -- insert the b.sequenceNumber in the attnSeqTable if not already there
  IF attnCount # 0 -- see if an entry is already there
    THEN
    FOR i IN (0..attnCount] DO
      IF attnSeqTable[i] = b.sequenceNumber THEN
        BEGIN alreadyThere ← TRUE; EXIT; END;
      ENDLOOP;
  IF NOT alreadyThere THEN -- because attnCount = 0 or because really not there!
    BEGIN
    IF (advanceB ← MaybeGetFreeReceiveSppBufferFromPool[pool]) = NIL THEN
      RETURN[FALSE]; -- cop out
    attnSeqTable[i] ← b.sequenceNumber;
    attnCount ← attnCount + 1;
    -- make a copy and then enqueue it
    to ← [@advanceB.checksum, 0, b.oisPktLength];
    from ← [@b.checksum, 0, b.oisPktLength];
    nBytes ← ByteBlt.ByteBlt[to: to, from: from];
    EnqueueSpp[@inAttnQueue, advanceB];
    NOTIFY newInputAttn;
    END;
  IF doStats THEN StatIncr[statAttentionsReceived];
  RETURN[TRUE];
  END; -- AttentionPacketProcessed

-- This procedure updates the attnSeqTable when the nextInputSeq is updated.
-- That is to say, we are removing any entries in the attnSeqTable that we plan
-- to ack, the next time we send out a packet.

UpdateAttnSeqTable: INTERNAL PROCEDURE =
  BEGIN
  i: CARDINAL ← 1;
  DO
    IF i > attnCount THEN EXIT;
    IF LOOPHOLE[(attnSeqTable[i] - nextInputSeq), INTEGER] < 0 THEN
      BEGIN
      attnSeqTable[i] ← attnSeqTable[attnCount];
      attnSeqTable[attnCount] ← 0;
      attnCount ← attnCount - 1;
      END
    ELSE i ← i + 1;
```



```
    ENDLOOP;
  END; -- UpdateAttnSeqTable

-- This procedure examines an error protocol packet.

GotErrorPacket: ENTRY PROCEDURE [b: OISCPDefs.OisBuffer] =
  BEGIN
  ENABLE UNWIND => NULL;
  IF doStats THEN StatIncr[statErrorPacketsReceived];
  OISCPDefs.ReturnReceiveOisBufferToPool[pool, b];
  END; -- GotErrorPacket

-- Cool Procedures

-- This procedure sets the wait time.

SetWaitTime: ENTRY PROCEDURE [time: WaitTime] =
  BEGIN waitPulses ← MilliSecondsToPulses[time]; END; -- SetWaitTime

-- This procedure returns the local and remote address of this packet stream.

FindAddresses: ENTRY PROCEDURE RETURNS [local, remote: OisAddress] =
  BEGIN local ← localAddr; remote ← remoteAddr; END; -- FindAddresses

-- This procedure returns the number of data bytes that can fit into an sppBuffer.

GetSenderSizeLimit: ENTRY PROCEDURE RETURNS [CARDINAL] =
  BEGIN RETURN[maxOisPktLength - bytesPerSequencedPktHeader]; END;
-- GetSenderSizeLimit



-- This procedure atempts to establish the local end of the connection
-- to the incoming packet.

EstablishThisConnection: INTERNAL PROCEDURE [b: SppBuffer]
  RETURNS [returnAnAck, giveBackToSocket: BOOLEAN] =
  BEGIN
  returnAnAck ← FALSE;
  giveBackToSocket ← TRUE;
  -- If state = activeEstablish then master/slave response when sockets are completely
  -- specified. If state = waitEstablish then other end initiates and knows all about us.
  IF b.source.host = remoteAddr.host AND b.source.socket = remoteAddr.socket AND
    b.destinationConnectionID = localConnectionID AND b.sourceConnectionID #
    unknownConnID AND b.sequenceNumber = 0 THEN
    BEGIN
    remoteConnectionID ← b.sourceConnectionID;
    maxOutputSeq ← b.allocationNumber;
    state ← established;
    IF b.sendAck THEN returnAnAck ← TRUE;
    IF NOT b.systemPacket THEN
      BEGIN
      -- if the attention bit is set then we try to process the attention, and only
      -- if we were able to do so do we put the packet on the inOrderQueue.
      IF ~b.attention OR (b.attention AND AttentionPacketProcessed[b]) THEN
        BEGIN
        nextInputSeq ← nextInputSeq + 1;
        EnqueueSpp[@inOrderQueue, b];
        giveBackToSocket ← FALSE;
```

```
      IF attnCount # 0 THEN UpdateAttnSeqTable[];
      -- remove knowledge of old attns
      state ← open;
      NOTIFY inOrderQueueNotEmpty;
      END;
    END
  ELSE IF doStats THEN StatIncr[statSystemPacketsReceived];
  NOTIFY connectionIsEstablished;
  END
  -- If state = activeEstablish then simultaneous establishment when sockets are
  -- completely specified. If state = waitEstablish then other end initiates and knows
  -- all about us except our connection ID.

ELSE
  IF b.source.host = remoteAddr.host AND b.source.socket = remoteAddr.socket
    AND b.destinationConnectionID = unknownConnID AND b.sourceConnectionID #
    unknownConnID AND b.sequenceNumber = 0 THEN
    BEGIN
    remoteConnectionID ← b.sourceConnectionID;
    maxOutputSeq ← b.allocationNumber;
    IF state = waitEstablish THEN returnAnAck ← TRUE;
    state ← established;
    IF b.sendAck THEN returnAnAck ← TRUE;
    IF doStats AND b.systemPacket THEN StatIncr[statSystemPacketsReceived];
    NOTIFY connectionIsEstablished;
    END
    -- If state = activeEstablish then response from well known socket at which
    -- there is a server process

  ELSE
    IF state = activeEstablish AND
      (b.source.host = remoteAddr.host AND b.source.socket #
        remoteAddr.socket) AND b.destinationConnectionID = localConnectionID
      AND b.sourceConnectionID # unknownConnID AND b.sequenceNumber = 0 THEN
      BEGIN
      remoteAddr.socket ← b.source.socket;
      remoteConnectionID ← b.sourceConnectionID;
      maxOutputSeq ← b.allocationNumber;
      state ← established;
      IF b.sendAck THEN returnAnAck ← TRUE;
      IF NOT b.systemPacket THEN
        BEGIN
        -- if the attention bit is set then we try to process the attention, and only
        -- if we were able to do so do we put the packet on the inOrderQueue.
        IF ~b.attention OR (b.attention AND AttentionPacketProcessed[b]) THEN
          BEGIN
          nextInputSeq ← nextInputSeq + 1;
          EnqueueSpp[@inOrderQueue, b];
          giveBackToSocket ← FALSE;
          IF attnCount # 0 THEN UpdateAttnSeqTable[];
          -- remove knowledge of old attns
          state ← open;
          NOTIFY inOrderQueueNotEmpty;
          END;
        END
      ELSE IF doStats THEN StatIncr[statSystemPacketsReceived];
      NOTIFY connectionIsEstablished;
      END
      -- mismatched IDs
```



```
    ELSE IF doStats THEN StatIncr[statPacketsRejectedBadID];
  IF returnAnAck AND doStats THEN StatIncr[statAckRequestsReceived];
  END; -- EstablishThisConnection

-- This procedure causes the active establishement of the connection. A system packet
-- is transmitted to the remote end, and then this procedure casues the caller to hang
-- for a certain maximum time in which it is hoped that the connection has become
-- established owing to the receipt of an appropriate packet from the remote end.

ActivelyEstablish: PROCEDURE =
  BEGIN
  currentPulse, startPulse: LONG CARDINAL;

  WaitUntilActivelyEstablished: ENTRY PROCEDURE RETURNS [BOOLEAN] = INLINE
    BEGIN
    ENABLE UNWIND => NULL;
    WAIT connectionIsEstablished;
    IF state = established OR state = open THEN RETURN[TRUE]
    ELSE
      BEGIN
      IF state = suspended THEN RETURN WITH ERROR ConnectionFailed[whyFailed];
      currentPulse ← GetClock[];
      IF (currentPulse - startPulse) >= waitPulses THEN
        BEGIN SIGNAL ConnectionFailed[timeout]; startPulse ← GetClock[]; END;
      RETURN[FALSE];
      END;
    END; -- WaitUntilActivelyEstablished

  startPulse ← GetClock[];
  DO
    SendSystemPacket[TRUE];
    -- waits for 2 seconds to see if the connection is established
    IF WaitUntilActivelyEstablished[] -- this returns or raises a SIGNAL -- THEN
      RETURN;
    ENDLOOP;
  END; -- ActivelyEstablish

-- This procedure waits for the remote end to initiate connection establishement.

WaitUntilEstablished: ENTRY PROCEDURE =
  BEGIN
  ENABLE UNWIND => NULL;
  currentPulse, startPulse: LONG CARDINAL;
  startPulse ← GetClock[];
  DO
    -- waits for 2 seconds each time around
    WAIT connectionIsEstablished;
    IF state = established OR state = open THEN RETURN
    ELSE
      BEGIN
      currentPulse ← GetClock[];
      IF (currentPulse - startPulse) >= waitPulses THEN
        BEGIN SIGNAL ConnectionFailed[timeout]; startPulse ← GetClock[]; END;
      END;
    ENDLOOP;
  END; -- WaitUntilEstablished

-- This procedure destroys the packet stream. The procedure changes the state of
-- the packet stream, NOTIFYs the Retransmitter and Receiver to self destruct,
-- and then waits until this happens. When this condition is satisfied, the socket
```

```
-- channel is deleted and, the packet stream data structures cleaned up.

Delete: PUBLIC PROCEDURE =
  BEGIN
  DeleteActivate[];
  Socket.Abort[SocketInternal.SocketHandleToChannelHandle[socketCH]];
  JOIN retransmitterFork;
  JOIN receiverFork;
  DeleteCleanup[];
  Socket.Delete[SocketInternal.SocketHandleToChannelHandle[socketCH]];
  Runtime.SelfDestruct[];
  END; -- Delete

DeleteActivate: ENTRY PROCEDURE = INLINE
  BEGIN
  state ← terminating;
  pleaseStop ← TRUE;
  NOTIFY retransmitterWakeupTimer;
  NOTIFY newInputAttn;
  END; -- DeleteActivate

DeleteCleanup: ENTRY PROCEDURE = INLINE
  BEGIN
  ENABLE UNWIND => NULL;
  IF sentBuffer # NIL THEN
    OISCPDefs.ReturnSendSppBufferToPool[pool, sentBuffer];
  BufferDefs.QueueCleanup[@inOrderQueue];
  BufferDefs.QueueCleanup[@inAttnQueue];
  BufferDefs.QueueCleanup[@outOfOrderQueue];
  BufferDefs.QueueCleanup[@tempQueue];
  BufferDefs.QueueCleanup[@sentQueue];
  END; -- DeleteCleanup

pulsesPerMilliSecond: LONG CARDINAL ← LOOPHOLE[System.MicrosecondsToPulses[
  1000], LONG CARDINAL];
oneMsecOfPulses: CARDINAL ← Inline.LowHalf[pulsesPerMilliSecond];
MilliSecondsToPulses: PROCEDURE [ms: LONG CARDINAL]
  RETURNS [pulses: LONG CARDINAL] =
  BEGIN
  -- we must be carefull about multiplication overflow since milliSeconds must be
  -- converted to microSeconds
  IF ms >= LAST[LONG CARDINAL]/1000 -- overflow condition
     THEN
    IF ms >= LAST[LONG CARDINAL]/pulsesPerMilliSecond -- ms is out of range
       THEN pulses ← LAST[LONG CARDINAL]
    ELSE pulses ← ms*pulsesPerMilliSecond -- close guestimation

  ELSE pulses ← LOOPHOLE[System.MicrosecondsToPulses[1000*ms], LONG CARDINAL];
  END; -- end MilliSecondsToPulses


-- initialization (Cool)
-- initialize connection control parameters

-- This initializes the necessary data structures for a packet stream instance.
-- It fills in all the necessary parameters and creates two processes that look
-- after the well being of this packet stream. One is a Retransmitter, that periodically
-- checks the sentQueue to see if there is anything unacknowledged, and the other
-- is the Receiver, which sees if there are any incoming packets on the
```



```
-- corresponding socket.

  state ← unestablished;
  whySuspended ← notSuspended;
  stateBeforeSuspension ← unestablished;
  Process.SetTimeout[@connectionIsEstablished, Process.MsecToTicks[2000]];
  socketCH ← SocketInternal.CreateInternal[
    localAddr, normal, sendBufs, receiveBufs];
  pool ← ps.pool ← SocketInternal.GetBufferPool[socketCH];
  localAddr ← Socket.GetStatus[
    SocketInternal.SocketHandleToChannelHandle[socketCH]].localAddr;
  IF remoteConnectionID # unknownConnID THEN state ← established;
  nextInputSeq ← unackedOutputSeq ← nextOutputSeq ← 0;
  maxInputSeq ← nextInputSeq + defaultRcvAllocation - 1;
  maxOutputSeq ← nextOutputSeq + defaultSendAllocation - 1;
  Process.SetTimeout[@newAllocation, Process.MsecToTicks[1000]];
  sendAnAck ← FALSE;
  maxOisPktLength ← maxBytesPerOisPkt;
  Process.SetTimeout[@newInputAttn, Process.MsecToTicks[1000]];
  FOR attnCount IN (0..maxPendingAttns] DO attnSeqTable[attnCount] ← 0; ENDLOOP;
  attnCount ← 0;
  sentBuffer ← NIL;
  BufferDefs.QueueInitialize[@inOrderQueue];
  BufferDefs.QueueInitialize[@inAttnQueue];
  BufferDefs.QueueInitialize[@outOfOrderQueue];
  BufferDefs.QueueInitialize[@tempQueue];
  BufferDefs.QueueInitialize[@sentQueue];
  Process.SetTimeout[@inOrderQueueNotEmpty, Process.MsecToTicks[1000]];
  ackRequestPulse ← lastProbePulse ← lastDelayCalculationPulse ←
    lastPacketReceivedPulse ← GetClock[];
  probeCounter ← 0;
  dataPacketRetransmitPulses ← MilliSecondsToPulses[
    initialDataPacketRetransmitTime]; -- msecs converted to Pulses
  probeRetransmitPulses ← MilliSecondsToPulses[
    probeMultiplier*initialDataPacketRetransmitTime + 40];
  -- msecs converted to Pulses
  delaySum ← 0;
  delayCount ← 0;
  normalRetransUpdates ← doubleRetransUpdates ← 0; -- **temp
  waitPulses ← MilliSecondsToPulses[timeout]; -- msecs converted to Pulses
  pleaseStop ← FALSE;
  Process.SetTimeout[@letClientRun, Process.MsecToTicks[250]];
  Process.SetTimeout[
    @retransmitterWakeupTimer, Process.MsecToTicks[initialRetransmitterTime]];
  -- This baroque code is to ensure that the state is not looked at by any of the other
  -- processes until this initialization code has finished examining this variable without
  -- entering the monitor.
  IF state = unestablished THEN
    BEGIN
    IF establish THEN
      BEGIN
      state ← activeEstablish;
      retransmitterFork ← FORK Retransmitter[];
      receiverFork ← FORK Receiver[];
      Process.Yield[];
      ActivelyEstablish[];
      END
    ELSE
```

```
      BEGIN
      state ← waitEstablish;
      retransmitterFork ← FORK Retransmitter[];
      receiverFork ← FORK Receiver[];
      Process.Yield[];
      WaitUntilEstablished[];
      END;
    END
  ELSE
    BEGIN
    SendSystemPacket[FALSE]; -- gratuitous ack in case we are a server agent now.
    -- Retransmissions of this gratuitous packet will ocurr in the nature of probes.
    retransmitterFork ← FORK Retransmitter[];
    receiverFork ← FORK Receiver[];
    Process.Yield[];
    END;
  RETURN[@ps, remoteAddr, remoteConnectionID];
  END. -- of PacketStreamInstance module
```

LOG

Time: May 9, 1978 3:39 PM By: Dalal Action: created file.
Time: October 11, 1978 4:25 PM By: Dalal Action: fixed CR 20.75.
Time: February 7, 1979 4:55 PM By: Dalal Action: conversion to Mesa 5.0.
Time: July 13, 1979 12:54 PM By: Dalal Action: fixed nested monitor problems.
Time: April 14, 1980 6:03 PM By: BLyon Action: Uses new Level1 stuff.
Time: June 13, 1980 3:50 PM By: Garlick Action: Added adaptive retransmission stuff.
Time: June 27, 1980 4:00 PM 3:50 PM By: Garlick Action: Changed to EXPORTED type ChannelH
***andle.*
Time: August 6, 1980 9:44 AM 3:50 PM By: Garlick Action: Made adaptive retransmission stuff s
***marter for the case with long delays (like phone net). It now uses something closer to the meas*
***ured delay for that case, rather than the delay times a large multiplier (used in the Ethernet case*
***because the measured standard deviation is large compared to delay). Also put a 20 sec ceiling*
***on the retransmit timeout.*
Time: September 11, 1980 4:59 PM By: Garlick Action: When passive end was sending a system
***packet to complete a connection, we weren't prepared for that packet getting lost. Fixed GotSe*
***quencedPacket to detect that case and send a system packet from passive side. (Second time t*
***his was fixed. There has always been a misaligned IF/THEN in this code.)*
Time: September 15, 1980 4:54 PM By: Garlick Action: 1.) shortened retransmissionsBeforeGive
***Up, probesBeforeGiveUp, and inactiveConnectionTime. 2.) implemented propogation of phone*
***network errors.*
Time: September 16, 1980 10:05 AM By: Garlick Action: Fixed zeroing of probeCounter.
Time: September 26, 1980 2:11 PM By: Garlick Action: Reduced flinging of probes by the passive
***receiver. It now considers the last time it saw a packet as a sign of activity, rather than the last ti*
***me it saw an ack (for something new).*
Time: October 2, 1980 3:15 PM By: Garlick Action: Added direct notification of the WaitForAttenti
***on condition variable to get the client out faster.*
Time: October 3, 1980 1:15 PM By: Garlick Action: Added seeting of ack req field if a packet is ret
***ransmitted more than retransmissionsBeforeAskForAck -1 times.*
Time: October 9, 1980 12:01 PM By: BLyon Action: Converted time unit from milliseconds to Syst
***em.Pulses.*
Time: October 12, 1980 5:31 PM By: BLyon Action: Added propogation of noAnswerOrBusy.

```
-- PacketStreamMgr.mesa (last edited by: BLyon on: October 1, 1980  10:46 AM)
-- Function: The implementation module for the manager of Pilot Packet Streams.

DIRECTORY
  CommunicationInternal: FROM "CommunicationInternal", -- EXPORTS
  DriverDefs USING [doDebug, Glitch],
  NetworkStream: FROM "NetworkStream", -- EXPORTS
  PacketStream USING [
    OisConnectionID, FailureReason, OisAddress, Handle, SuspendReason,
    uniqueAddress, unknownConnID, WaitTime],
  PacketStreamInstance USING [Delete],
  Router USING [AssignDestinationRelativeOisAddress],
  Runtime USING [GlobalFrame];

PacketStreamMgr: MONITOR
  IMPORTS Runtime, DriverDefs, pktStrmInst: PacketStreamInstance, Router
  EXPORTS CommunicationInternal, NetworkStream, PacketStream =
  BEGIN OPEN Router, PacketStream;

  -- These variables must eventually live in outerspace so that multiple MDSs
  -- access the same packet stream variables.  The module in the primary MDS will
  -- perform the initialization of the "globals", while the others will not.

  -- This module is a monitor in order to protect the value of spareConnectionID and the
  -- ConnectionTable.  Some day the connection IDs in use will be remembered across wrap
  -- around.

  primaryMDS: PUBLIC BOOLEAN ← TRUE; -- we are in the primary MDS.
  initialSpareConnectionID: OisConnectionID = [500];
  -- monitor data
  -- connectionID parameters
  spareConnectionID: OisConnectionID;
  -- connectionTable parameters and types
  ConnectionTable: ARRAY (0..maxConnectionNumber] OF ConnectionTableEntry;
  maxConnectionNumber: CARDINAL = 20;
  index: CARDINAL ← 0;
  ConnectionTableEntry: TYPE = RECORD [
    rAddr: OisAddress, rConnID: OisConnectionID];

  -- various Glitches.
  ConnectionTableFull: ERROR = CODE;
  NotInConnectionTable: ERROR = CODE;

  ConnectionSuspended: PUBLIC ERROR [why: SuspendReason] = CODE;
  ConnectionFailed: PUBLIC SIGNAL [why: FailureReason] = CODE;
  AttentionTimeout: PUBLIC ERROR = CODE;

  -- Cool Procedures

  -- This procedure creates a packet stream instance.  The local and remote addresses
  -- must be correctly specified, or else we pick defaults when possible.
  Make: PUBLIC PROCEDURE [
    local, remote: OisAddress, localConnID, remoteConnID: OisConnectionID,
    establishConnection: BOOLEAN, timeout: WaitTime]
    RETURNS [psH: PacketStream.Handle] =
    BEGIN
    estdRemoteAddr: OisAddress;
    estdRemoteConnID: OisConnectionID;
```



```
    newPktStrmInst: POINTER TO FRAME[PacketStreamInstance] ← NEW pktStrmInst;
    BEGIN
    ENABLE UNWIND => newPktStrmInst.Delete[];
    -- If we do not have a local address we pick one with a network number that is a
    -- good way to get to the destinationand therefore a good way to get back to us!
    IF local = uniqueAddress THEN
      local ← Router.AssignDestinationRelativeOisAddress[remote.net];
    IF localConnID = unknownConnID THEN localConnID ← GetUniqueConnectionID[];
    [psH, estdRemoteAddr, estdRemoteConnID] ← START newPktStrmInst[
      local, remote, localConnID, remoteConnID, establishConnection, timeout];
    InsertIntoConnectionTable[estdRemoteAddr, estdRemoteConnID];
    END;
    END; -- Create

  -- This procedure deletes the specified packet stream instance.

  Destroy: PUBLIC PROCEDURE [psH: PacketStream.Handle] =
    BEGIN
    oldPktStrmInst: POINTER TO FRAME[PacketStreamInstance] ←
      LOOPHOLE[Runtime.GlobalFrame[psH.get], POINTER];
    remote: OisAddress ← oldPktStrmInst.remoteAddr;
    remoteConnID: OisConnectionID ← oldPktStrmInst.remoteConnectionID;
    oldPktStrmInst.Delete[];
    RemoveFromConnectionTable[remote, remoteConnID];
    END; -- Delete

  -- This procedure returns a unique connection ID.  Some day the active connection IDs
  -- in use will be kept around, so that on wrap around an unused ID will be assigned.

  GetUniqueConnectionID: ENTRY PROCEDURE RETURNS [iD: OisConnectionID] =
    BEGIN
    iD ← spareConnectionID;
    IF (spareConnectionID ← [spareConnectionID + 1]) = OisConnectionID[0] THEN
      spareConnectionID ← initialSpareConnectionID;
    END; -- GetUniqueConnectionID

  -- This procedure inserts a connection into the connectionTable.

  InsertIntoConnectionTable: PUBLIC ENTRY PROCEDURE [
    remote: OisAddress, remoteConnID: OisConnectionID] =
    BEGIN
    IF (index ← index + 1) > maxConnectionNumber THEN
      IF DriverDefs.doDebug THEN DriverDefs.Glitch[ConnectionTableFull];
    ConnectionTable[index] ← ConnectionTableEntry[remote, remoteConnID];
    END; -- InsertIntoConnectionTable

  -- This procedure removes a connection from the connectionTable.

  RemoveFromConnectionTable: PUBLIC ENTRY PROCEDURE [
    remote: OisAddress, remoteConnID: OisConnectionID] =
    BEGIN
    i: CARDINAL;
    FOR i IN (0..index] DO
      IF ConnectionTable[i].rAddr.host = remote.host AND ConnectionTable[
        i].rAddr.socket = remote.socket AND ConnectionTable[i].rConnID =
        remoteConnID THEN
        BEGIN
        ConnectionTable[i] ← ConnectionTable[index];
        index ← index - 1;
```

```
        RETURN;
        END;
      ENDLOOP;
    IF DriverDefs.doDebug THEN DriverDefs.Glitch[NotInConnectionTable];
    END; -- RemoveFromConnectionTable

  -- This procedure checks if there is a similar connection in the connectionTable.

  ConnectionAlreadyThere: PUBLIC ENTRY PROCEDURE [
    remote: OisAddress, remoteConnID: OisConnectionID] RETURNS [BOOLEAN] =
    BEGIN
    i: CARDINAL;
    FOR i IN (0..index] DO
      IF ConnectionTable[i].rAddr.host = remote.host AND ConnectionTable[
        i].rAddr.socket = remote.socket AND ConnectionTable[i].rConnID =
        remoteConnID THEN RETURN[TRUE];
      ENDLOOP;
    RETURN[FALSE];
    END; -- ConnectionAlreadyThere

  -- initialization

  IF primaryMDS THEN spareConnectionID ← initialSpareConnectionID;

  END. -- of PacketStreamMgr module
```

LOG

Time: January 26, 1980 1:35 PM By: Dalal Action: created file.
Time: January 26, 1980 1:36 PM By: Dalal Action: split NetworkStreamMgr into two.
Time: April 15, 1980 10:25 AM By: BLyon Action: uses new Ivel1 stuff.

```
-- PhoneNetworkDriver.mesa (last edited by: Garlick on: October 12, 1980 9:18 PM)

DIRECTORY
  BufferDefs: FROM "BufferDefs" USING [
    Buffer, OisBuffer, PupBuffer, RppBuffer, BufferType],
  CommUtilDefs: FROM "CommUtilDefs" USING [GetEthernetHostNumber],
  Dialup: FROM "Dialup" USING [Dial, RetryCount],
  DriverDefs: FROM "DriverDefs" USING [
    NetworkObject, Network, PutOnGlobalDoneQueue, DriverXmitStatus, doStats,
    AddDeviceToChain, GetInputBuffer, PutOnGlobalInputQueue, ReturnFreeBuffer,
    RemoveDeviceFromChain],
  DriverTypes: FROM "DriverTypes" USING [
    Byte, phoneEncapsulationOffset, phoneEncapsulationBytes],
  Environment: FROM "Environment" USING [Block],
  HalfDuplex: FROM "HalfDuplex" USING [
    Initialize, Destroy, WaitToSend, SendCompleted, CheckForTurnAround],
  OISCPTypes: FROM "OISCPTypes" USING [OisNetID, unknownNetID, phoneNetID],
  OISTransporter: FROM "OISTransporter",
  PhoneNetwork: FROM "PhoneNetwork" USING [FindPhonePath, UnknownPath],
  Process: FROM "Process" USING [SetTimeout, MsecToTicks],
  PupTypes: FROM "PupTypes" USING [PupHostID, PupErrorCode],
  RS232C: FROM "RS232C" USING [
    ChannelHandle, CompletionHandle, PhysicalRecord, PhysicalRecordHandle,
    TransferStatus, DeviceStatus, LineSpeed, Get, Put, TransferWait, TransmitNow,
    GetStatus, StatusWait, SetParameter, Suspend, Restart, ChannelSuspended],
  RS232CManager: FROM "RS232CManager" USING [
    NetAccess, CommParamObject, CommParamHandle, CommDuplex],
  SpecialSystem: FROM "SpecialSystem" USING [
    GetProcessorID, HostNumber, ProcessorID, nullProcessorID,
    broadcastHostNumber];

PhoneNetworkDriver: MONITOR
  IMPORTS
    CommUtilDefs, Dialup, DriverDefs, HalfDuplex, PhoneNetwork, Process, RS232C,
    SpecialSystem
  EXPORTS OISTransporter
  SHARES BufferDefs =
  BEGIN
  -- various definitions
  NetworkState: TYPE = {available, unavailable};
  LineState: TYPE = {closed, active};
  ConnectionStatus: TYPE = {successful, modemDown, channelPreempted};
  CreatePhonePathOutcome: TYPE = {
    success, busyOrNoAnswer, noDialer, noTranslation, dialerError};
  CheckPathOutcome: TYPE = {
    success, noTranslation, busyOrNoAnswer, noDialer, dialerError, circuitInUse};
  StatsRecord: TYPE = RECORD [
    pktsSent, pktsReceived, pktsRejected, notTheCurrentPath, noPathSendNoNet,
      busyOrNoAnswer, noTranslationForAddress, sendErrorBadStatus,
      noPathSendLineDown, rcvErrorDataLost, rcvErrorChecksum, rcvErrorNoGet,
      rcvErrorUnknown, rcvErrorFrameTimeout, rcvDeviceError, bytesSent,
      bytesReceived, dialError, dsrDropped: CARDINAL];
  -- state things
  lineState: LineState; -- becomes active when DSR comes up
  networkState: NetworkState;
  -- becomes available after we tell router about ourselves; unavailable when channel deleted (incl
**udes preempted)
  sendRecProcessesActive: BOOLEAN;
```



```
  -- FALSE means they cannot be JOINed during cleanup
  dialing: BOOLEAN; -- Receiver process waits for this
  awaitingLineUpAfterDial: BOOLEAN; -- Receiver process waits for this
  dialComplete: CONDITION;
  -- current channel usage
  channelHandle: RS232C.ChannelHandle;
  currentPathSystemElement: SpecialSystem.HostNumber;
  mySystemElement: SpecialSystem.ProcessorID;
  duplexity: RS232CManager.CommDuplex;
  modemSpeed: RS232C.LineSpeed;
  lineMode: RS232CManager.NetAccess;
  autoDial: BOOLEAN;
  dialRetries: Dialup.RetryCount;
  -- output queue
  headOutputBuffer, tailOutputBuffer: BufferDefs.Buffer;
  packetToSend: CONDITION;
  -- phone network object for Router
  phoneNetObject: DriverDefs.NetworkObject ←
    [next: NIL, decapsulateBuffer: TypeOfBuffer, encapsulatePup: PupFrameIt,
      encapsulateRpp: RppFrameIt, encapsulateOis: OISFrameIt,
      sendBuffer: EnqueueSend, forwardBuffer: ForwardFrame, -- rejects
      activateDriver: ActivateTransporter, deactivateDriver: KillTransporter,
      deleteDriver: KillTransporter, interrupt: InterruptNop,
      -- this gets locked! (a crok)
      index:, device: phonenet, alive: TRUE, speed: 1, buffers: 0, spare:,
      netNumber:, hostNumber:, pupStats: StatsNop, stats: NIL];
  phoneNetwork: DriverDefs.Network ← @phoneNetObject;
  -- process handles
  senderProcess: PROCESS;
  receiverProcess: PROCESS;
  statusWaitProcess: PROCESS;
  -- stats
  statsRec: StatsRecord;
  -- constants
  noError: DriverTypes.Byte = 0B;
  noPathError: DriverTypes.Byte = 100B;
  noTranslationError: DriverTypes.Byte = 101B;
  circuitInUseError: DriverTypes.Byte = 102B;
  noDialerError: DriverTypes.Byte = 103B;
  dialerHardwareError: DriverTypes.Byte = 104B;
  noAnswerOrBusyError: DriverTypes.Byte = 104B;
  dialNetNumber: OISCPTypes.OisNetID = OISCPTypes.phoneNetID;
  outstandingGets: CARDINAL = 2;
  -- determines the amount of Channel receives for multiple buffering
  -- signals and errors
  -- ********** Initialization/Termination (from RS232CManager) **********
  Initialize: PUBLIC PROCEDURE [
    chHandle: RS232C.ChannelHandle, commParams: RS232CManager.CommParamObject] =
    -- start the Transporter with the given params
    BEGIN
    -- locals
    -- initialize globals
    lineState ← closed; -- becomes active when DSR comes up
    networkState ← unavailable;
    -- becomes available after we tell router about ourselves
    sendRecProcessesActive ← FALSE;
    awaitingLineUpAfterDial ← FALSE;
    channelHandle ← chHandle;
```

```
    duplexity ← commParams.duplex;
    modemSpeed ← commParams.lineSpeed;
    headOutputBuffer ← tailOutputBuffer ← NIL;
    -- stats
    IF DriverDefs.doStats THEN
      statsRec ← [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
    -- get our own host number, initialize remote ID
    mySystemElement ← SpecialSystem.GetProcessorID[];
    phoneNetwork.hostNumber ← CommUtilDefs.GetEthernetHostNumber[];
    currentPathSystemElement ← SpecialSystem.nullProcessorID;
    -- decode Communication parameters
    WITH commParams SELECT FROM
      directConn =>
        BEGIN
        phoneNetwork.netNumber ← OISCPTypes.unknownNetID;
        lineMode ← directConn;
        END;
      dialConn =>
        BEGIN
        phoneNetwork.netNumber ← dialNetNumber;
        autoDial ← (dialMode = auto);
        dialRetries ← retryCount;
        lineMode ← dialConn;
        END;
      ENDCASE;
    -- set RS-232-C parameters that matter (note: bitSynchronous has mostly defaults)
    RS232C.SetParameter[channelHandle, [dataTerminalReady[TRUE]]];
    -- tell modem we are ready to boogie
    networkState ← available; -- allow sends from Router
    -- start status change watch process (which will start the receiver process and any half-duplex)
    statusWaitProcess ← FORK StatusChangeWait[];
    -- tell Router about ourselves
    DriverDefs.AddDeviceToChain[phoneNetwork, 0];
    END;

  Destroy: PUBLIC PROCEDURE [chHandle: RS232C.ChannelHandle] =
    -- stop everything so we don't make any more channel calls
    BEGIN
    -- locals
    RS232C.Suspend[channelHandle, all];
    JOIN statusWaitProcess; -- sender and receiver JOINED by status wait guy

    END;
  -- ********** Encapsulation **********

  FrameAny: PROCEDURE [buffer: BufferDefs.Buffer] =
    -- fill buffer with level 0 header stuff
    BEGIN
    -- locals
    -- encapsulation offset with no HDLC framing should be 3, with HDLC framing should be 1
    buffer.encapsulation ←
      [phonenet[
        framing0: noError, framing1: 0, framing2: 0, framing3: 0, framing4: 0,
        framing5: 0, recognition: 0, -- for auto-recog
        pnType: oisPhonePacket, pnSrcID: mySystemElement]];
    END;

  OISFrameIt: PROCEDURE [
```



```
    buffer: BufferDefs.OisBuffer, systemElem: SpecialSystem.HostNumber] =
    -- fill buffer with OIS level 0 header stuff
    BEGIN
    -- locals
    errorByte: DriverTypes.Byte;
    -- check if we are talking to the system element (may cause dialing)
    errorByte ←
      SELECT CheckPath[systemElem] FROM
        success => noError,
        busyOrNoAnswer => noAnswerOrBusyError,
        noTranslation => noTranslationError,
        dialerError => dialerHardwareError,
        noDialer => noDialerError,
        circuitInUse => circuitInUseError,
        ENDCASE => noPathError;
    IF errorByte # noError THEN
      BEGIN -- set error in encapsulation field
      buffer.encapsulation.framing0 ← errorByte;
      IF DriverDefs.doStats THEN
        SELECT errorByte FROM
          = noAnswerOrBusyError => StatIncr[@statsRec.busyOrNoAnswer];
          = noTranslationError => StatIncr[@statsRec.noTranslationForAddress];
          = circuitInUseError => StatIncr[@statsRec.notTheCurrentPath];
          ENDCASE;
      RETURN;
      END;
    FrameAny[buffer];
    buffer.encapsulation.pnType ← oisPhonePacket;
    buffer.length ←
      (buffer.oisPktLength + 1 + DriverTypes.phoneEncapsulationBytes)/2;
    -- note that length is in words. Since SPP protocol does checksums on words, it expects the dr
**iver to pad, i.e. huge hack in OISCP.

    END;

  PupFrameIt: PROCEDURE [
    buffer: BufferDefs.PupBuffer, systemElem: PupTypes.PupHostID] =
    -- fill buffer with Pup level 0 header stuff
    BEGIN
    -- locals
    FrameAny[buffer];
    buffer.encapsulation.pnType ← pupPhonePacket;
    buffer.length ←
      (buffer.pupLength + 1 + DriverTypes.phoneEncapsulationBytes)/2;
    -- note that length is in rounded-up words, because higher-level checksum on words

    END;

  RppFrameIt: PROCEDURE [
    buffer: BufferDefs.RppBuffer, systemElem: PupTypes.PupHostID] =
    LOOPHOLE[PupFrameIt];
  TypeOfBuffer: PROCEDURE [buffer: BufferDefs.Buffer]
    RETURNS [type: BufferDefs.BufferType] =
    -- determine buffer type
    BEGIN
    SELECT buffer.encapsulation.pnType FROM
      oisPhonePacket => type ← ois;
      pupPhonePacket => type ← pup;
```

```
    ENDCASE =>
      BEGIN
      type ← rejected;
      IF DriverDefs.doStats THEN StatIncr[@statsRec.pktsRejected];
      END;
  END;
-- ********** Packet Transport / Sending **********

EnqueueSend: ENTRY PROCEDURE [buffer: BufferDefs.Buffer] =
  -- queue the buffer for output
  BEGIN
  -- locals
  errorByte: DriverTypes.Byte ← buffer.encapsulation.framing0;
  xmitStatus: DriverDefs.DriverXmitStatus;
  -- check for error states (some set in encapsulation routine)
  IF (lineState = closed) OR (networkState = unavailable) OR
    (errorByte # noError) THEN
    BEGIN
    IF networkState = unavailable THEN xmitStatus ← noRouteToNetwork
    ELSE
      IF ~awaitingLineUpAfterDial THEN
        BEGIN
        xmitStatus ←
          SELECT errorByte FROM
            = noPathError => noRouteToNetwork,
            = noAnswerOrBusyError => noAnswerOrBusy,
            = noTranslationError => noTranslationForDestination,
            = dialerHardwareError => dialerHardwareProblem,
            = noDialerError => noDialingHardware,
            = circuitInUseError => circuitInUse,
          ENDCASE =>
            IF lineState = closed THEN circuitNotReady ELSE noRouteToNetwork;
        END
      ELSE xmitStatus ← goodCompletion; -- drop on the floor waiting for DSR
    buffer.status ← LOOPHOLE[xmitStatus];
    DriverDefs.PutOnGlobalDoneQueue[buffer];
    IF DriverDefs.doStats THEN
      BEGIN
      IF lineState = closed THEN StatIncr[@statsRec.noPathSendLineDown];
      IF networkState = unavailable THEN StatIncr[@statsRec.noPathSendNoNet];
      END;
    RETURN;
    END;
  IF headOutputBuffer = NIL THEN headOutputBuffer ← tailOutputBuffer ← buffer
  ELSE BEGIN tailOutputBuffer.next ← buffer; tailOutputBuffer ← buffer; END;
  NOTIFY packetToSend;
  END;

Sender: PROCEDURE =
  -- process that waits for things to send
  BEGIN
  -- locals
  b: BufferDefs.Buffer;
  AwaitPacketToSend: ENTRY PROCEDURE =
    -- wait for a non-empty queue
    BEGIN
    UNTIL (headOutputBuffer # NIL) OR ~sendRecProcessesActive DO
      WAIT packetToSend; ENDLOOP;
    END;
  UNTIL ~sendRecProcessesActive DO
```



```
    AwaitPacketToSend[]; SendFrame[DequeueSend[]]; ENDLOOP;
  -- clear any remaining things to send and go away
  UNTIL (b ← DequeueSend[]) = NIL DO
    DriverDefs.PutOnGlobalDoneQueue[b]; ENDLOOP;
  END;

NotifySenderToGoAway: ENTRY PROCEDURE =
  -- make Sender wake up
  BEGIN NOTIFY packetToSend; END;

DequeueSend: ENTRY PROCEDURE RETURNS [buffer: BufferDefs.Buffer] =
  -- dequeue the buffer for output
  BEGIN
  -- locals
  IF headOutputBuffer = NIL THEN buffer ← NIL
  ELSE BEGIN buffer ← headOutputBuffer; headOutputBuffer ← buffer.next; END;
  END;

SendFrame: PROCEDURE [buffer: BufferDefs.Buffer] =
  -- build and Put a frame to the channel
  BEGIN
  -- locals
  complHandle: RS232C.CompletionHandle;
  rec: RS232C.PhysicalRecord ←
    [header: [NIL, 0, 0], body:, trailer: [NIL, 0, 0]];
  IF buffer = NIL THEN RETURN;
  -- FIll Channel Physical record
  rec.body.blockPointer ←
    @buffer.encapsulation + DriverTypes.phoneEncapsulationOffset;
  -- word boundary
  rec.body.startIndex ← 0;
  rec.body.stopIndexPlusOne ← buffer.length*2; -- even bytes
  -- if half duplex, set RTS and await CTS
  IF duplexity = half THEN HalfDuplex.WaitToSend[];
  -- do the Put
  complHandle ← RS232C.Put[
    channelHandle, @rec !
    RS232C.ChannelSuspended =>
      BEGIN -- someone wants us to go away
      PreemptedSending[buffer];
      IF duplexity = half THEN HalfDuplex.SendCompleted[FALSE];
      GOTO returnPlace;
      END];
  -- wait for completion
  SendWait[buffer, complHandle];
  IF duplexity = half THEN HalfDuplex.SendCompleted[(headOutputBuffer # NIL)];
  -- give a chance for line turn-around and say if we have more to send

  EXITS returnPlace => NULL;
  END;

SendWait: PROCEDURE [
  buffer: BufferDefs.Buffer, complete: RS232C.CompletionHandle] =
  -- waits for Put to complete
  BEGIN
  -- locals
  bytes: CARDINAL;
  xferstatus: RS232C.TransferStatus;
```

```
    -- wait for completion
    [bytes, xferstatus] ← RS232C.TransmitNow[channelHandle, complete];
    -- fill status, bytes in buffer
    buffer.status ← TranslateChannelStatus[xferstatus];
    buffer.length ← bytes;
    -- stats
    IF DriverDefs.doStats THEN
      IF xferstatus = success THEN
        BEGIN
        StatIncr[@statsRec.pktsSent];
        StatBump[@statsRec.bytesSent, bytes];
        END
      ELSE StatIncr[@statsRec.sendErrorBadStatus];
    -- call completion procedure
    DriverDefs.PutOnGlobalDoneQueue[buffer];
    END;

  TranslateChannelStatus: PRIVATE PROCEDURE [chStatus: RS232C.TransferStatus]
    RETURNS [oisStatus: CARDINAL] =
    -- translate channel status to driver xmit status
    BEGIN -- need more definitive codes ???????
    -- locals
    frameStatus: DriverDefs.DriverXmitStatus;
    SELECT chStatus FROM
      success => frameStatus ← goodCompletion;
      aborted => frameStatus ← aborted;
      ENDCASE => frameStatus ← hardwareProblem;
    oisStatus ← LOOPHOLE[frameStatus, CARDINAL];
    END;

  PreemptedSending: PRIVATE PROCEDURE [buffer: BufferDefs.Buffer] =
    -- called if channel ripped away due to preemption (by Channel Mgr)
    BEGIN
    networkUnavailableStatus: DriverDefs.DriverXmitStatus ← aborted;
    buffer.status ← LOOPHOLE[networkUnavailableStatus, CARDINAL];
    --we may not be a network driver if preemption reported in Receiver or StatusWait, but we can
**still return the buffer (Yogen says)
    DriverDefs.PutOnGlobalDoneQueue[buffer];
    END;

  ForwardFrame: PROCEDURE [buffer: BufferDefs.Buffer]
    RETURNS [PupTypes.PupErrorCode] =
    -- reject forward attempt
    BEGIN RETURN[gatewayResourceLimitsPupErrorCode]; END;
  -- ********** Packet Transport / Receiving **********

  Receiver: PROCEDURE =
    -- process that receives frames and notifies OIS
    BEGIN
    -- locals
    recArray: ARRAY [0..outstandingGets) OF RS232C.PhysicalRecord;
    bufferArray: ARRAY [0..outstandingGets) OF BufferDefs.Buffer;
    complArray: ARRAY [0..outstandingGets) OF RS232C.CompletionHandle;
    preemptedArray: ARRAY [0..outstandingGets) OF BOOLEAN;
    someoneGotPreempted: BOOLEAN ← FALSE;
    i: CARDINAL;
    -- wait for dialing if necessary
    WaitDialComplete[];
```



```
    -- perform some Gets for multiple buffering
    IF lineState = active AND networkState = available THEN
      FOR i IN [0..outstandingGets) DO
        recArray[i] ← [[NIL, 0, 0], [NIL, 0, 0], [NIL, 0, 0]];
        bufferArray[i] ← DriverDefs.GetInputBuffer[];
        -- gets buffer.length (words) includes encaps.
        -- perform receive
        [preemptedArray[i], complArray[i]] ← DoGet[@recArray[i], bufferArray[i]];
        IF (someoneGotPreempted ← preemptedArray[i]) THEN EXIT;
        ENDLOOP;
    UNTIL (lineState = closed) OR (networkState = unavailable) OR
      someoneGotPreempted DO
      -- get frames, pass to OIS
      FOR i IN [0..outstandingGets) DO
        -- wait for a completion and give to Router or discard
        AwaitAndDisposeOfFrame[complArray[i], bufferArray[i]];
        -- get next buffer
        bufferArray[i] ← DriverDefs.GetInputBuffer[];
        -- gets buffer.length (words) includes encaps.
        -- perform next receive
        [preemptedArray[i], complArray[i]] ← DoGet[@recArray[i], bufferArray[i]];
        IF (someoneGotPreempted ← preemptedArray[i]) THEN EXIT;
        ENDLOOP;
      ENDLOOP; -- of UNTIL
    -- wait for Get completion of those not preempted
    FOR i IN [0..outstandingGets) DO
      IF ~preemptedArray[i] THEN
        AwaitAndDisposeOfFrame[complArray[i], bufferArray[i]];
      ENDLOOP;
    END;

  DoGet: PRIVATE PROCEDURE [
    recHandle: RS232C.PhysicalRecordHandle, buffer: BufferDefs.Buffer]
    RETURNS [preempted: BOOLEAN, complHandle: RS232C.CompletionHandle] =
    -- perform the channel Get and watch for preemption
    -- assumes record header and trailer nil
    BEGIN
    -- locals
    preempted ← FALSE;
    -- fill physical record (it is being reused)
    -- clear encapsulation words ?????
    recHandle.body ←
      [blockPointer: @buffer.encapsulation + DriverTypes.phoneEncapsulationOffset,
        startIndex: 0,
        stopIndexPlusOne:
        (buffer.length - DriverTypes.phoneEncapsulationOffset)*2];
    -- perform receive
    -- need more error handling here ????
    complHandle ← RS232C.Get[
      channelHandle, recHandle !
      RS232C.ChannelSuspended =>
        BEGIN
        DriverDefs.ReturnFreeBuffer[buffer]; -- return unused buffer
        preempted ← TRUE;
        CONTINUE;
        END];
    END;
```

```
AwaitAndDisposeOfFrame: PROCEDURE [
  complHandle: RS232C.CompletionHandle, buffer: BufferDefs.Buffer] =
  -- wait for get to complete and either hand to router or throw away
  BEGIN
  -- locals
  transferStatus: RS232C.TransferStatus;
  globalStatus: RS232C.DeviceStatus;
  bytes: CARDINAL;
  statsPtr: POINTER TO CARDINAL;
  checksum: CARDINAL = 2;
  -- wait for completion (no errors can occur here)
  [bytes, transferStatus] ← RS232C.TransferWait[channelHandle, complHandle];
  IF transferStatus = success THEN
    BEGIN -- pass only good frames to Router
    -- set byte count for consistency check in decapsulation
    buffer.length ← bytes - checksum;
    -- add network object to buffer
    buffer.network ← phoneNetwork;
    buffer.device ← phonenet;
    -- check contents of header and trailer
    currentPathSystemElement ← buffer.encapsulation.pnSrcID;
    IF (duplexity = half) AND HalfDuplex.CheckForTurnAround[buffer] THEN
      BEGIN -- throw buffer away
      DriverDefs.ReturnFreeBuffer[buffer];
      RETURN;
      END;
    -- notify OIS
    DriverDefs.PutOnGlobalInputQueue[buffer];
    -- stats
    IF DriverDefs.doStats THEN
      BEGIN
      StatIncr[@statsRec.pktsReceived];
      StatBump[@statsRec.bytesReceived, bytes - checksum];
      END;
    END
  ELSE
    BEGIN -- bad frame => free the buffer
    DriverDefs.ReturnFreeBuffer[buffer];
    IF DriverDefs.doStats AND (transferStatus # aborted) THEN
      BEGIN
      statsPtr ←
        SELECT transferStatus FROM
          dataLost => @statsRec.rcvErrorDataLost,
          checksumError => @statsRec.rcvErrorChecksum,
          frameTimeout => @statsRec.rcvErrorFrameTimeout,
          deviceError => @statsRec.rcvDeviceError,
          ENDCASE => @statsRec.rcvErrorUnknown;
      StatIncr[statsPtr];
      END;
    IF transferStatus = deviceError THEN
      BEGIN
      globalStatus ← RS232C.GetStatus[channelHandle];
      IF globalStatus.dataLost THEN -- clear the data lost latch bit
        BEGIN
        RS232C.SetParameter[
          channelHandle, [latchBitClear[globalStatus]] !
          RS232C.ChannelSuspended => CONTINUE];
        IF DriverDefs.doStats THEN StatIncr[@statsRec.rcvErrorNoGet];
        END;
```



```
      END;
    END;
  END;
-- ********** Driver Management (by Router) **********

ActivateTransporter: PROCEDURE =
  -- call from Router when we add ourselves as network driver
  BEGIN
  -- locals

  END;

KillTransporter: PROCEDURE =
  -- called by Router when we removed ourselves as a network driver
  BEGIN
  -- locals

  END;

StatsNop: PROCEDURE [buffer: BufferDefs.PupBuffer, network: DriverDefs.Network]
  RETURNS [BOOLEAN] =
  -- NOP
  BEGIN
  -- locals
  RETURN[FALSE];
  END;

InterruptNop: PROCEDURE =
  -- NOP
  BEGIN
  -- locals

  END;
-- ********** Connection Management **********

CheckPath: PRIVATE PROCEDURE [systemElem: SpecialSystem.HostNumber]
  RETURNS [outcome: CheckPathOutcome] =
  -- create new path if necessary & auto-dial mode
  BEGIN
  -- locals
  IF lineMode = directConn OR ~autoDial THEN RETURN[success];
  IF lineState = closed THEN
    BEGIN -- must create auto-dialed path
    SELECT CreatePhonePath[systemElem] FROM
      success =>
        BEGIN currentPathSystemElement ← systemElem; RETURN[success]; END;
      noTranslation => RETURN[noTranslation];
      busyOrNoAnswer => RETURN[busyOrNoAnswer];
      dialerError => RETURN[dialerError];
      noDialer => RETURN[noDialer];
      ENDCASE => RETURN[dialerError];
    END
  ELSE
    BEGIN -- check if same as previous or probing (uses broadcast processor ID)
    IF systemElem = currentPathSystemElement OR systemElem =
      SpecialSystem.broadcastHostNumber --probing-- THEN RETURN[success]
    ELSE RETURN[circuitInUse];
    END;
  END;
```

```
CreatePhonePath: PROCEDURE [systemElem: SpecialSystem.HostNumber]
  RETURNS [outcome: CreatePhonePathOutcome] =
  -- dial thru the auto-dial hardware
  BEGIN
  -- locals
  commParamHandle: RS232CManager.CommParamHandle;
  phoneNumber: STRING;
  dialerZero: CARDINAL = 0;
  -- get phone number, etc.
  [commParamHandle, phoneNumber] ← PhoneNetwork.FindPhonePath[
    systemElem ! PhoneNetwork.UnknownPath => GOTO noTranslation];
  -- set params that may be different (ignore for now, use global ones)
  -- dial it
  dialing ← TRUE;
  outcome ←
    SELECT Dialup.Dial[dialerZero, phoneNumber, dialRetries] FROM
      success => success,
      failure => busyOrNoAnswer,
      dialerNotPresent => noDialer,
      ENDCASE => dialerError;
  IF ~(outcome = success) THEN
    IF DriverDefs.doStats THEN StatIncr[@statsRec.dialError];
  NotifyDialComplete[(outcome = success)];
  RETURN[outcome];
  EXITS noTranslation => RETURN[noTranslation];
  END;

NotifyDialComplete: ENTRY PROCEDURE [successfulDial: BOOLEAN] =
  -- tell Receiver that it can receive now (in case it was created because DSR came up)
  BEGIN
  dialing ← FALSE;
  awaitingLineUpAfterDial ← successfulDial;
  NOTIFY dialComplete;
  END;

WaitDialComplete: ENTRY PROCEDURE =
  -- wait for dialing and resetting of line params (if not RS366)
  BEGIN UNTIL ~dialing DO WAIT dialComplete; ENDLOOP; END;

StatusChangeWait: PRIVATE PROCEDURE =
  -- a process to wait for an abnormal status change
  -- this process must terminate if channel goes away
  BEGIN
  --locals
  newStatus: RS232C.DeviceStatus;
  BEGIN
  DO
    -- until channel is preempted
    -- wait for complete connection (may be complete already)
    newStatus ← RS232C.GetStatus[
      channelHandle ! RS232C.ChannelSuspended => GOTO preempted];
    UNTIL newStatus.dataSetReady DO
      newStatus ← RS232C.StatusWait[
        channelHandle, newStatus ! RS232C.ChannelSuspended => GOTO preempted];
      ENDLOOP;
    lineState ← active;
    CreateSendReceiveProcesses[];
    -- await line drop or preemption
    UNTIL ProcessStatusChange[newStatus] = modemDown DO
```



```
      newStatus ← RS232C.StatusWait[
        channelHandle, newStatus ! RS232C.ChannelSuspended => GOTO preempted];
      ENDLOOP;
    IF DriverDefs.doStats THEN StatIncr[@statsRec.dsrDropped];
    -- abort the channel to reach consistent state (make Receiver go away)
    lineState ← closed;
    RS232C.Suspend[channelHandle, input];
    RS232C.Suspend[channelHandle, output];
    RemoveSendReceiveProcesses[];
    RS232C.Restart[channelHandle, input];
    RS232C.Restart[channelHandle, output];
    currentPathSystemElement ← SpecialSystem.nullProcessorID;
    -- current guy is nobody

    ENDLOOP;
  EXITS preempted => BEGIN PreemptCleanup[]; RemoveSendReceiveProcesses[]; END;
  END;
  END;

ProcessStatusChange: PRIVATE ENTRY PROCEDURE [status: RS232C.DeviceStatus]
  RETURNS [ConnectionStatus] =
  -- check line status; determine if transient
  BEGIN
  --locals
  timeOut: CONDITION;
  dsrRecovTimeInSecs: CARDINAL = 2; -- secs to wait for DataSetReady recovery
  loopWait: CARDINAL ← 500; -- msecs between checking for DataSetReady
  loopCount: CARDINAL ← dsrRecovTimeInSecs*1000/loopWait;
  ourWaits: CARDINAL ← 0;
  newStatus: RS232C.DeviceStatus;
  -- check for lost line
  IF ~status.dataSetReady THEN
    BEGIN -- wait a reasonable time for it to come back without hogging
    UNTIL ourWaits > loopCount DO
      Process.SetTimeout[@timeOut, Process.MsecToTicks[loopWait]];
      WAIT timeOut;
      newStatus ← RS232C.GetStatus[
        channelHandle ! RS232C.ChannelSuspended => GOTO fatalPlace];
      IF newStatus.dataSetReady THEN RETURN[successful];
      ourWaits ← ourWaits + 1;
      REPEAT fatalPlace => RETURN[channelPreempted];
      ENDLOOP;
    RETURN[modemDown];
    END
  ELSE RETURN[successful];
  END;

CreateSendReceiveProcesses: PROCEDURE =
  -- start the receiver
  BEGIN
  awaitingLineUpAfterDial ← FALSE;
  sendRecProcessesActive ← TRUE;
  receiverProcess ← FORK Receiver;
  senderProcess ← FORK Sender;
  IF duplexity = half THEN
    HalfDuplex.Initialize[channelHandle, modemSpeed, mySystemElement];
  END;
```

```
  RemoveSendReceiveProcesses: PROCEDURE =
    -- bring back the receiver
    BEGIN
    IF sendRecProcessesActive THEN
      BEGIN
      sendRecProcessesActive ← FALSE;
      NotifySenderToGoAway[];
      IF duplexity = half THEN HalfDuplex.Destroy[];
      JOIN receiverProcess;
      JOIN senderProcess;
      END;
    END;

  PreemptCleanup: ENTRY PROCEDURE =
    -- called if channel ripped away due to preemption (by Channel Mgr)
    BEGIN
    IF networkState = available THEN
      BEGIN -- inform our cohorts
      networkState ← unavailable; -- inform other processes
      DriverDefs.RemoveDeviceFromChain[phoneNetwork];
      END;
    END;
  -- ***************** Statistics *****************

  StatIncr: PROCEDURE [counter: POINTER TO CARDINAL] =
    -- add one to counter
    BEGIN
    -- locals
    counter↑ ← (counter↑ + 1) MOD (LAST[CARDINAL] - 1);
    END;

  StatBump: PROCEDURE [counter: POINTER TO CARDINAL, bumpAmount: CARDINAL] =
    -- add bumpAmount to counter; if bumpAmount < 10000, there will never be overflow
    BEGIN
    -- locals
    counter↑ ← (counter↑ + bumpAmount) MOD (LAST[CARDINAL] - 10000);
    END;
  -- ***************** Main Program *****************

  END.
```

Issues to address in future:
1. Don't hog the output buffers so other media drivers (especially Ethernets) get their share. Eithe
***r drop packets on the floor or send back a flow control status to the router.*
2. Keep the queue length for half duplex and for # 1. Can use the MoreToSend packet-type to tell
***the other end to hurry with the line turn-around.*

LOG
Time: August 2, 1978 10:04 AM By: Garlick Action: Created file
Time: January 22, 1980 1:29 PM By: Garlick Action: fixed CreatePhonePath to
***call RS232C.SetLineType after dialing.*
Time: January 25, 1980 5:13 PM By: Garlick Action: fixed initialization of curre
***ntPathSystemElement in InitOisTransporter and StatusChangeWait.*
Time: January 29, 1980 9:48 AM By: Garlick Action: made Receiver wait for di
***aling and all associated line resetting before performing Gets, since SetLineType does an implic*
***it abort of the channel.*
Time: January 29, 1980 11:49 AM By: Garlick Action: added padding of OISCP
***packet so that we always send integral words. This is a compatibility service (hack) provided by*
***all drivers.*
Time: January 30, 1980 6:27 PM By: Garlick Action: made changes for compat

***ibility with new SpecialSystem, new RS232C and RS366, and new PhoneNetwork.*
Time: March 17, 1980 9:28 AM By: Garlick Action: in CheckPath, allows a broadcastPro
***cessorID at any time. Allows flexibility in our probe protocol.*
Time: June 11, 1980 2:49 PM By: McJones Action: broadcastProcessorID = >broadcast
***HostNumber.*
Time: July 2, 1980 12:01 PM By: Garlick Action: changed name of module to PhoneN
***etworkDriver.*
Time: July 7, 1980 1:24 AM By: Garlick Action: made termination compatible with n
***ew RS232C changes.*
Time: July 18, 1980 2:17 PM By: Garlick Action: added half-duplex support and a que
***ued sending interface.*
Time: August 5, 1980 6:39 PM By: Garlick Action: 1.) changed EnqueueSend to not rep
***ort noRouteToNetwork when awaiting line up after a dial. 2.) made compatible with new PhoneN*
***etwork (that uses HostNumbers). 3.) added a Destroy and renamed InitOisTransporter to be Initi*
***alize. Now we JOIN the StatusWait process.*
Time: August 6, 1980 8:57 AM By: Garlick Action: No longer get a net number for a lea
***sed line; rather use OISCPTypes.unknownNetID. Use OISCPTypes.phoneNetID for the phone n*
***etwork number instead of 40B.*
Time: August 11, 1980 3:18 PM By: Garlick Action: Changed order of JOINs and deletin
***g HalfDuplex guy.*
Time: September 15, 1980 9:26 AM By: Garlick Action: Implemented return of ma
***ny new error codes to be returned when sending.*
Time: September 18, 1980 5:17 PM. By: BLyon Action: added buffers & spare to phoneNetO
***bject.*
Time: September 19, 1980 5:22 PM By: Garlick Action: fixed bugs in error codes.
***CircuitNotReady overriding other errors.*
Time: October 12, 1980 9:18 PM By: Garlick Action: added implementtation of
***DriverDefs XmitStatus[noAnswerOrBusy].*

```
-- PhoneNetworkImpl.mesa (last edited by: Garlick on: August 5, 1980 5:44 PM) --
DIRECTORY
  PhoneNetwork USING [AccessAddress],
  Heap USING [CreateMDS, MakeMDSNode, FreeMDSNode, MakeMDSString, FreeMDSString],
  RS232CManager USING [CommParamHandle, CommParamObject],
  SpecialSystem USING [HostNumber];

PhoneNetworkImpl: MONITOR IMPORTS Heap EXPORTS PhoneNetwork =
  BEGIN
  -- types
  PathList: TYPE = RECORD [head, tail: PathElemHandle];
  PathElemHandle: TYPE = POINTER TO PathListElem;
  PathListElem: TYPE = RECORD [
    previous: PathElemHandle,
    next: PathElemHandle,
    systemElemID: SpecialSystem.HostNumber,
    lineParams: RS232CManager.CommParamHandle,
    phoneNo: PhoneNetwork.AccessAddress];
  -- writeable data
  myHeap: MDSZone;
  -- data structure for keeping path info
  phoneBook: PathList ← [NIL, NIL];
  -- constants
  heapPages: CARDINAL = 1;
  heapThreshold: CARDINAL = 4; -- min block size
  heapPageIncrement: CARDINAL = 1;
  -- errors and signals
  UnknownPath: PUBLIC ERROR = CODE;
  -- ********Phone Network Path Definition********
  FindPhonePath: PUBLIC ENTRY PROCEDURE [
    OISUniqueAddress: SpecialSystem.HostNumber]
    RETURNS [RS232CManager.CommParamHandle, PhoneNetwork.AccessAddress] =
    -- search path table
    BEGIN
    ENABLE UNWIND => NULL; -- release monitor
    -- local
    pathElem: PathElemHandle;
    -- start at head
    pathElem ← phoneBook.head;
    UNTIL pathElem = NIL DO
      IF pathElem.systemElemID = OISUniqueAddress THEN
        RETURN[pathElem.lineParams, pathElem.phoneNo];
      pathElem ← pathElem.next;
      ENDLOOP;
    ERROR UnknownPath;
    END; -- disable UNWIND

  KnowAboutPhonePath: PUBLIC ENTRY PROCEDURE [
    OISUniqueAddress: SpecialSystem.HostNumber,
    commParamHandle: RS232CManager.CommParamHandle,
    phoneNumber: PhoneNetwork.AccessAddress] =
    -- add to path table
    BEGIN
    -- locals
    lineParams: RS232CManager.CommParamHandle;
    accessString: STRING;
    pathElem: PathElemHandle;
    BEGIN
    ENABLE UNWIND => NULL; -- release monitor
```



```
    -- allocate, copy CommParamObject, AccessAddress
    lineParams ← Heap.MakeMDSNode[myHeap, SIZE[RS232CManager.CommParamObject]];
    lineParams↑ ← commParamHandle↑;
    accessString ← Heap.MakeMDSString[myHeap, phoneNumber.length];
    accessString↑ ← [length: 0, maxlength: phoneNumber.length, text:];
    AppendString[accessString, phoneNumber];
    -- allocate table entry
    pathElem ← Heap.MakeMDSNode[myHeap, SIZE[PathListElem]];
    END; -- disable UNWIND
    pathElem↑ ← PathListElem[
      previous: NIL, next: phoneBook.head, systemElemID: OISUniqueAddress,
      lineParams: lineParams, phoneNo: accessString];
    -- Add to head of phone list
    IF phoneBook.head # NIL THEN phoneBook.head.previous ← pathElem;
    phoneBook.head ← pathElem;
    IF phoneBook.tail = NIL THEN phoneBook.tail ← pathElem;
    END;

  ForgetAboutPhonePath: PUBLIC ENTRY PROCEDURE [
    OISUniqueAddress: SpecialSystem.HostNumber] =
    -- delete the path from path table
    BEGIN
    ENABLE UNWIND => NULL; -- release monitor
    -- local
    pathElem: PathElemHandle;
    -- if active, what will happen? Nothing, transporter will handle it
    pathElem ← phoneBook.head;
    UNTIL pathElem = NIL DO
      -- we only look at the processor ID of the networkAddress
      IF pathElem.systemElemID = OISUniqueAddress THEN
        BEGIN -- delete it
        IF pathElem = phoneBook.head THEN phoneBook.head ← pathElem.next
        ELSE pathElem.previous.next ← pathElem.next;
        IF pathElem = phoneBook.tail THEN phoneBook.tail ← pathElem.previous
        ELSE pathElem.next.previous ← pathElem.previous;
        -- free everything
        Heap.FreeMDSNode[myHeap, pathElem.lineParams];
        Heap.FreeMDSString[myHeap, pathElem.phoneNo];
        Heap.FreeMDSNode[myHeap, pathElem];
        RETURN;
        END;
      pathElem ← pathElem.next;
      ENDLOOP;
    ERROR UnknownPath;
    END; -- disable UNWIND
  -- ********String Support********

  AppendString: PROCEDURE [to: STRING, from: STRING] =
    BEGIN -- assumes to string has room!!
    i, j, n: CARDINAL;
    IF to = NIL OR from = NIL THEN RETURN;
    n ← MIN[from.length, LOOPHOLE[to.maxlength - to.length, CARDINAL]];
    i ← to.length;
    j ← 0;
    WHILE j < n DO to[i] ← from[j]; i ← i + 1; j ← j + 1; ENDLOOP;
    to.length ← i;
    RETURN
    END;
```

```
  -- MAIN PROGRAM --
  -- create a heap

  myHeap ← Heap.CreateMDS[
    initial: heapPages, increment: heapPageIncrement, threshold: heapThreshold];
  END.
LOG
Time: January 30, 1980  3:58 PM          By: Garlick          Action: Created file from subset o
**f RS232CManagerImpl.
Time: July 3, 1980  9:33 AM      By: Garlick          Action: Stopped using RS232CHeap.  Starte
**d using Heap.
Time: January 30, 1980  3:58 PM          By: Garlick          Action: Stopped using RS232CHe
**ap.  Started using Heap.
Time: July 21, 1980  12:26 PM    By: McJones          Action: Heap.Create + GetMDSHandle = >Cr
**eateMDS
Time: August 5, 1980  5:43 PM    By: Garlick          Action: Changed all SpecialSystem.Process
**orIDs to HostNumber so we could accept broadcastHostNumber.
```

```
-- RouterImpl.mesa (last edited by: BLyon on: September 18, 1980 3:35 PM)
-- Function: The implementation module for the Pilot OISCP Router switch.

DIRECTORY
  BufferDefs: FROM "BufferDefs", -- only for SHARES
  ByteBlt USING [ByteBlt],
  Checksums USING [SetChecksum, TestChecksum],
  CommunicationInternal: FROM "CommunicationInternal", -- EXPORTS
  CommUtilDefs USING [CopyLong],
  DriverDefs USING [
    doDebug, doStats, doStorms, Glitch, Network, RouterObject, SetOisRouter],
  Environment USING [Block],
  Inline USING [LowHalf],
  OISCPDefs USING [
    DequeueOis, EnqueueOis, GetFreeOisBuffer, OisAddress, OisBuffer, OisNetID,
    OisHostID, OisSocketID, MaybeGetFreeReceiveOisBufferFromPool],
  OISCPTypes USING [
    allHostIDs, bytesPerOisPktHeader, bytesPerOisPktText, maxBytesPerOisPkt,
    routingInformationSocket, unknownNetID],
  Router USING [
    AddNetwork, FindNetworkAndTransmit, FindDestinationRelativeNetID,
    ForwardPacket, RemoveNetwork, RoutingInformationPacket, RoutingTableOn,
    RoutingTableOff, SocketTable, StateChanged, XmitStatus],
  Socket USING [PhysicalRecord, OISPktHeaderRecord],
  SocketInternal USING [SocketHandle],
  SpecialCommunication USING [RoutersFunction],
  SpecialSystem USING [GetProcessorID],
  StatsDefs USING [StatIncr],
  System USING [GetClockPulses];

RouterImpl: MONITOR LOCKS socketRouterLock
  IMPORTS
    ByteBlt, Checksums, CommUtilDefs, DriverDefs, Inline, OISCPDefs, Router,
    SpecialSystem, StatsDefs, System
  EXPORTS CommunicationInternal, OISCPDefs, Router
  SHARES BufferDefs =
  BEGIN OPEN DriverDefs, OISCPDefs, OISCPTypes, Socket, SocketInternal, StatsDefs;

  -- Many of these variables must eventually live in outerspace so that multiple MDSs
  -- access the same router variables. The modules in the primary MDS will have the
  -- proceses and will perform the initialization of the "globals", while the others will not.

  -- The lock covers both SocketImpl and RouterImpl, specifically the socket objects,
  -- the socket table, and spare socket ID counter. These will all live in outerspace.
  -- The monitor locks some variables on the global frame too. They can be MDS specific.

  -- switches and variables that don't change during execution unless for diagnostics
  primaryMDS: PUBLIC BOOLEAN ← TRUE; -- we are in the primary MDS.
  routersFunction: PUBLIC SpecialCommunication.RoutersFunction ← vanillaRouting;
  myHostID: OISCPDefs.OisHostID; -- processor ID of this system element
  checkIt: PUBLIC BOOLEAN ← TRUE; -- checksums on for everybody
  driverLoopback: BOOLEAN ← FALSE; -- loopback in router
  stormy: BOOLEAN ← FALSE; -- storms for debugging are not on

  initialSpareSocketID: OISCPDefs.OisSocketID = [1001];

  -- oiscp router object for the dispatcher. There will only be one for all MDSs, but it
  -- can live on all global frame instances or we can put it in hyperspace and have the
```



```
  -- dispatcher access it via a long pointer. The former is preferable.
  oiscpRouter: DriverDefs.RouterObject ←
    [input: LOOPHOLE[ReceivePacket], broadcast: LOOPHOLE[SendBroadcastPacket],
      addNetwork: Router.AddNetwork, removeNetwork: Router.RemoveNetwork,
      stateChanged: Router.StateChanged];

  -- monitor protected data.
  -- parameters for killing packets for debugging
  lightning: INTEGER ← 30;
  bolt: INTEGER ← 10;
  -- I think only the following need be in outerspace, accesible via a pointer since these
  -- will be touched by multiple MDS processes via their copy of RouterImpl and SocketImpl.
  socketRouterLock: PUBLIC MONITORLOCK;
  -- lock for the router and all the sockets.
  -- Some day the socketIDs in use will be remembered across wrap around, and maybe
  -- the rebooting of Pilot.
  spareSocketID: OISCPDefs.OisSocketID;
  -- socket table
  socketTable: Router.SocketTable;

  -- various Glitches generated by the router
  IllegalOisPktLength: ERROR = CODE;

  -- Cool Procedures

  -- This procedure assigns a temporary socket number in an OisAddress.
  -- Some day the active socket numbers in use will be kept around, so that on wrap
  -- around an unused number will be assigned. When the system element is connected to
  -- more than one network, this procedure must return a list of OisAddress.
  AssignOisAddress: PUBLIC ENTRY PROCEDURE
    RETURNS [localAddr: OISCPDefs.OisAddress] =
    BEGIN
    localAddr ← [net: unknownNetID, host: myHostID, socket: spareSocketID];
    IF (spareSocketID ← [spareSocketID + 1]) = OISCPDefs.OisSocketID[0] THEN
      spareSocketID ← initialSpareSocketID;
    END; -- AssignOisAddress

  -- This procedure is just like the previous one except that the network number is relative
  -- to the destination network. That is, we pick that one of our locally connected
  -- networks that is the best way to get to destNet, with the hope that it is the best
  -- way to get from destNet to us.

  AssignDestinationRelativeOisAddress: PUBLIC ENTRY PROCEDURE [destNet: OisNetID]
    RETURNS [localAddr: OISCPDefs.OisAddress] =
    BEGIN
    localAddr ←
      [net: Router.FindDestinationRelativeNetID[destNet], host: myHostID,
        socket: spareSocketID];
    IF (spareSocketID ← [spareSocketID + 1]) = OISCPDefs.OisSocketID[0] THEN
      spareSocketID ← initialSpareSocketID;
    END; -- AssignDestinationRelativeOisAddress

  -- This procedure tells the OISCP Router about a new socket.

  AddSocket: PUBLIC ENTRY PROCEDURE [sH: SocketHandle] =
    BEGIN
    -- add new socket to the head of the table
    sH.next ← socketTable.first;
```

```
  socketTable.first ← sH;
  socketTable.length ← socketTable.length + 1;
  END; -- AddSocket

-- This procedure removes a socket from the OISCP Router's tables.

RemoveSocket: PUBLIC ENTRY PROCEDURE [sH: SocketHandle] =
  BEGIN
  previousSH: SocketHandle;
  IF socketTable.first = sH THEN socketTable.first ← sH.next
  ELSE
    BEGIN
    previousSH ← socketTable.first;
    UNTIL previousSH = NIL DO
      IF previousSH.next = sH THEN BEGIN previousSH.next ← sH.next; EXIT; END;
      previousSH ← previousSH.next;
      ENDLOOP;
    END;
  socketTable.length ← socketTable.length - 1;
  END; -- RemoveSocket

-- This is not an entry procedure because we change it only for debugging and can
-- live with a race condition.

SetOisStormy: PUBLIC PROCEDURE [new: BOOLEAN] = BEGIN stormy ← new; END;
-- SetOisStormy

-- This is not an entry procedure because we change it only for debugging and can
-- live with a race condition.

SetOisCheckit: PUBLIC PROCEDURE [new: BOOLEAN] = BEGIN checkIt ← new; END;
-- SetOisCheckIt

-- This is not an entry procedure because we change it only for debugging and can
-- live with a race condition.

SetOisDriverLoopback: PUBLIC PROCEDURE [new: BOOLEAN] =
  BEGIN driverLoopback ← new; END; -- SetOisDriverLoopback

-- This procedure returns the processor ID of this machine.

FindMyHostID: PUBLIC PROCEDURE RETURNS [OisHostID] =
  BEGIN RETURN[myHostID]; END; -- FindMyHostID

GetOisPacketTextLength: PUBLIC PROCEDURE [b: OisBuffer] RETURNS [CARDINAL] =
  BEGIN RETURN[b.oisPktLength - bytesPerOisPktHeader]; END;
-- GetOisPacketTextLength

SetOisPacketTextLength: PUBLIC PROCEDURE [b: OisBuffer, len: CARDINAL] =
  BEGIN
  IF len IN [0..bytesPerOisPktText] THEN
    b.oisPktLength ← len + bytesPerOisPktHeader
  ELSE IF doDebug THEN Glitch[IllegalOisPktLength];
  END; -- SetOisPacketTextLength

SetOisPacketLength: PUBLIC PROCEDURE [b: OisBuffer, len: CARDINAL] =
  BEGIN
  IF len IN [bytesPerOisPktHeader..maxBytesPerOisPkt] THEN b.oisPktLength ← len
  ELSE IF doDebug THEN Glitch[IllegalOisPktLength];
```



```
  END; -- SetOisPacketLength

--GetDoStats: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
--BEGIN
--RETURN[doStats];
--END; -- -- GetDoStats

-- Hot Procedures

-- This procedure transmits a packet over a locally connected network.
-- The procedure assumes that all fields of the buffer have been filled in appropriately,
-- except the encapsulation and buffer length which depend on the network. If the
-- packet is destined for a local socket and we do NOT want to do loopback at the
-- driver or at the network, then it gets looped back here. Broadcast
-- packets are NOT delivered to the source socket.
-- Packets are no longer copied into system buffers.
-- The caller owns the buffer UNLESS it is a system buffer.

SendPacket: PUBLIC PROCEDURE [b: OisBuffer] =
  BEGIN

  IF doStorms AND stormy AND PacketHit[b] THEN RETURN; -- for debugging only

    IF b.destination.host = myHostID AND NOT driverLoopback THEN
    -- packet is for a local socket; broadcast packets are not delivered locally
    BEGIN
    IF NOT DeliveredToLocalSocket[b] THEN
      IF doStats THEN StatIncr[statJunkOisForUsNoLocalSocket]; -- no socket

    END
  ELSE
    -- packet is for a remote machine
    BEGIN
    -- If the buffer is a crate (the send data is not in the buffer) then copy the header
    -- info and data to a system buffer before transmission. If the destination net is
    -- inaccessible then we will have suffered the overhead of coputing a checksum,
    -- and maybe a copy, but we don't expect inaccessible nets that often.
    -- set the checksum if appropriate
    IF checkIt THEN Checksums.SetChecksum[b] ELSE b.checksum ← 177777B;
    IF (b.size # big) AND (b.userData # NIL) THEN
      BEGIN
      copiedB: OisBuffer ← GetFreeOisBuffer[];
      CommUtilDefs.CopyLong[
        from: @b.checksum, nwords: bytesPerOisPktHeader/2,
        to: @copiedB.checksum];
      CommUtilDefs.CopyLong[
        from: b.userData, nwords: (b.userDataLength + 1)/2,
        -- we take an extra byte if byte count is odd
        to: @copiedB.oisBody];
      b.status ← LOOPHOLE[Router.FindNetworkAndTransmit[copiedB]];
      END
    ELSE BEGIN [] ← Router.FindNetworkAndTransmit[b]; END;
    END;
  END; -- SendPacket

-- This procedure is called by the Dispatcher when it sees a valid ois packet in a system
-- buffer. The procedure checks the checksum field and then routes the packet to a
-- local socket. If this is an internetwork router, then the packets are forwarded over
-- a suitable network. We now own this packet.
```

```
ReceivePacket: PUBLIC PROCEDURE [b: OisBuffer] =
  BEGIN
  badChecksum: BOOLEAN;
  destHost: OisHostID ← b.destination.host;


  IF (badChecksum ← IF checkIt THEN NOT Checksums.TestChecksum[b] ELSE FALSE)
    THEN
    -- hardware says ok, but bad end-to-end software checksum.
    BEGIN
    IF doStats THEN
      BEGIN
      StatIncr[statReceivedBadOisChecksum];
      StatIncr[statOisDiscarded];
      END;
    b.requeueProcedure[b]; -- done with this packet
    RETURN;
    END;

  -- packet is in good shape, route it.

    IF doStorms AND stormy AND PacketHit[b] THEN
    BEGIN b.requeueProcedure[b]; RETURN; END; -- for debugging only

    IF destHost = myHostID OR destHost = allHostIDs THEN
    BEGIN -- incomming packet for us (we may have broadcast it)
    IF NOT DeliveredToLocalSocket[b] THEN
      BEGIN
      -- routing information socket is part of router in another module
      IF b.destination.socket = routingInformationSocket THEN
        Router.RoutingInformationPacket[b] -- we still own this packet!

      ELSE
        BEGIN -- packet for unknown socket
        IF doStats THEN
          IF destHost # allHostIDs THEN StatIncr[statJunkOisForUsNoLocalSocket]
          ELSE StatIncr[statJunkBroadcastOis];
        END;
      END;
    b.requeueProcedure[b]; -- done with this packet

    END
  ELSE
    IF routersFunction = interNetworkRouting THEN
      BEGIN
      Router.ForwardPacket[b]; -- dispatcher returns to system pool

      END
    ELSE
      BEGIN
      IF doStats THEN StatIncr[statOisDiscarded];
      b.requeueProcedure[b]; -- we got the packet when we shouldn't have!

      END;
  END; -- ReceivePacket

-- Hot, except should really not exist; therefor cold?
-- This procedure attempts to hit a packet with a bolt of lightening, and if it succeeds,
-- then it returns true else false. The caller dispenses with the buffer.
```



```
PacketHit: ENTRY PROCEDURE [b: OisBuffer] RETURNS [BOOLEAN] =
  BEGIN
  IF (lightning ← lightning + 1) > bolt OR lightning < 0 THEN
    BEGIN
    IF lightning > bolt THEN
      BEGIN
      IF bolt > 100 THEN
        BEGIN
        randLong: LONG CARDINAL ← System.GetClockPulses[];
        rand: CARDINAL ← Inline.LowHalf[randLong];
        lightning ← -INTEGER[rand MOD 20B];
        bolt ← 10;
        END
      ELSE BEGIN lightning ← 0; bolt ← bolt + 1; END;
      END;
    IF doStats THEN StatIncr[statZappedP];
    RETURN[TRUE];
    END
  ELSE RETURN[FALSE];
  END; -- PacketHit

-- This procedure finds a local socket object to deliver the packet to, and if it succeeds,
-- then it returns true else false. The caller dispenses with the buffer.

DeliveredToLocalSocket: ENTRY PROCEDURE [b: OisBuffer] RETURNS [BOOLEAN] =
  BEGIN
  ENABLE UNWIND => NULL;
  destSocket: OisSocketID ← b.destination.socket;
  sH: SocketHandle;
  -- find the correct socket for this packet
  FOR sH ← socketTable.first, sH.next UNTIL sH = NIL DO
    IF sH.localAddr.socket = destSocket THEN
      BEGIN EnqueueNewInput[sH, b]; RETURN[TRUE]; END;
    ENDLOOP;
  RETURN[FALSE];
  END; -- DeliveredToLocalSocket

-- This procedure enqueues a new input packet at the socket object.
-- The incoming buffer may be from a local socket, or from a network driver.
-- The caller of this routine returns this buffer appropriately.

EnqueueNewInput: INTERNAL PROCEDURE [sH: SocketHandle, b: OISCPDefs.OisBuffer] =
  INLINE
  BEGIN
  ENABLE UNWIND => NULL;
  LongPhysicalRecordHandle: TYPE = LONG POINTER TO Socket.PhysicalRecord;
  fromBlock, toBlock: Environment.Block;
  nBytes: CARDINAL;
  getBuffer: OisBuffer ←
    SELECT TRUE FROM
      sH.channelState = aborted => NIL,
      sH.bufferPoolType = normal => MaybeGetFreeReceiveOisBufferFromPool[
        @sH.poolObject],
      sH.bufferPoolType = crate => DequeueOis[@sH.pendingUserGetQueue],
      ENDCASE => NIL;
  -- Copy b into the first OisBuffer on the pendingGetQueue, if there is one.
  IF (getBuffer # NIL) THEN
    BEGIN
    getBuffer.status ← LOOPHOLE[Router.XmitStatus[goodCompletion]];
    -- assume always good
```

```
    -- perform the copy.  There are 4 cases
    IF getBuffer.size = user AND b.size = user THEN
      -- both getBuffer and b are crates
      BEGIN
      CommUtilDefs.CopyLong[
        -- copy the header
        from: @b.checksum, nwords: SIZE[OISPktHeaderRecord],
        to: @LOOPHOLE[getBuffer.userPtr, LongPhysicalRecordHandle].header];
      fromBlock ← [b.userData, 0, b.userDataLength];
      toBlock ← [getBuffer.userData, 0, getBuffer.userDataLength];
      nBytes ← ByteBlt.ByteBlt[toBlock, fromBlock];
      IF getBuffer.userDataLength < b.userDataLength THEN
        getBuffer.status ← LOOPHOLE[Router.XmitStatus[crateTooSmall]]
      END
    ELSE
      IF getBuffer.size = user AND b.size = big THEN
        -- getBuffer is a crate, b is a big buffer
        BEGIN
        CommUtilDefs.CopyLong[
          -- copy the header
          from: @b.checksum, nwords: SIZE[OISPktHeaderRecord],
          to: @LOOPHOLE[getBuffer.userPtr, LongPhysicalRecordHandle].header];
        fromBlock ← [@b.oisBody, 0, (b.oisPktLength - bytesPerOisPktHeader)];
        toBlock ← [getBuffer.userData, 0, getBuffer.userDataLength];
        nBytes ← ByteBlt.ByteBlt[toBlock, fromBlock];
        IF getBuffer.userDataLength < (b.oisPktLength - bytesPerOisPktHeader)
          THEN getBuffer.status ← LOOPHOLE[Router.XmitStatus[crateTooSmall]]
        END
      ELSE
        IF getBuffer.size = big AND b.size = user THEN
          -- getBuffer is a full sized OisBuffer, b is a crate.  No need to check sizes
          BEGIN
          CommUtilDefs.CopyLong[
            from: @b.checksum, nwords: SIZE[OISPktHeaderRecord],
            to: @getBuffer.checksum];
          fromBlock ← [b.userData, 0, b.userDataLength];
          toBlock ← [@getBuffer.oisBody, 0, bytesPerOisPktText];
          nBytes ← ByteBlt.ByteBlt[toBlock, fromBlock];
          END
        ELSE
          IF getBuffer.size = big AND b.size = big THEN
            -- getBuffer is full sized OisBuffer, therefore no need to check length
            CommUtilDefs.CopyLong[
              from: @b.checksum, nwords: (b.oisPktLength + 1)/2,
              to: @getBuffer.checksum];
    IF (sH.bufferPoolType = normal) THEN
      BEGIN
      EnqueueOis[@sH.completedReceiveQueue, getBuffer];
      BROADCAST sH.newReceiveInput;
      END
    ELSE
      BEGIN
      EnqueueOis[@sH.completedUserGetQueue, getBuffer];
      BROADCAST sH.newUserInput;
      END;
    END
  ELSE IF DriverDefs.doStats THEN StatsDefs.StatIncr[statOisInputQueueOverflow];
  -- funny name

  END; -- EnqueueNewInput
```



```
-- This procedure causes a broadcast packet to be sent out over the right network.

SendBroadcastPacket: PUBLIC PROCEDURE [b: OisBuffer] = BEGIN ERROR; END;
-- SendBroadcastPacket

--Cold Procedures

-- This procedure turns the router on.  The Communication software is written with the
-- idea of eventually turning the code on and off during execution, and so we should
-- get the latest values of netID and hostID when being turned back on.

OisRouterOn: PUBLIC PROCEDURE =
  BEGIN
  OisRouterActivate[];
  Router.RoutingTableOn[];
  DriverDefs.SetOisRouter[@oiscpRouter];
  END; -- OisRouterOn

OisRouterActivate: ENTRY PROCEDURE = INLINE
  BEGIN
  myHostID ← SpecialSystem.GetProcessorID[];
  IF primaryMDS THEN socketTable ← [length: 0, first: NIL];
  END; -- OisRouterActivate

OisRouterOff: PUBLIC PROCEDURE =
  BEGIN
  DriverDefs.SetOisRouter[NIL];
  OisRouterDeactivate[];
  Router.RoutingTableOff[];
  END; -- OisRouterOff

OisRouterDeactivate: ENTRY PROCEDURE = INLINE
  BEGIN
  -- cleanup the socket table.
  --IF primaryMDS THEN

  END; -- OisRouterDeactivate

--Cold
-- initialization

IF primaryMDS THEN spareSocketID ← initialSpareSocketID;

END. -- RouterImpl module.
```

LOG

Time: January 19, 1980 4:05 PM By: Dalal Action: Split OISCPRouter into two.
Time: January 21, 1980 6:07 PM By: Dalal Action: one lock for SocketImpl and RouterImpl.
Time: March 13, 1980 4:55 PM By: BLyon Action: modified SendPacket.
Time: March 18, 1980 4:09 PM By: BLyon Action: Modified EnqueueNewInput where it gets an in
**put buffer.
Time: May 12, 1980 6:38 PM By: BLyon Action: Put checksum into microcode (switched order pa
**rameters too).
Time: May 16, 1980 10:07 AM PM By: BLyon Action: Removed all ShortenPointer to allow multipl
**e MDS.
Time: June 30, 1980 1:02 PM By: BLyon Action: Checkit init to FALSE instead of TRUE..
Time: July 22, 1980 11:03 AM By: BLyon Action: Checkit changed back to TRUE; checksums stuf

***I put in seperate modules; we now receive out own broadcasts.*
Time: August 1, 1980 1:29 PM By: BLyon Action: replaced internetRouter by routersFunction.
Time: September 13, 1980 6:15 PM By: HGM Action: Add StateChanged.
Time: September 18, 1980 3:34 PM By: BLyon Action: AssignOisAddress puts unknownNetID in n
***etwork field rather than primaryNetID.*

```
-- RoutingTableImpl.mesa (last edited by: BLvon on: September 19, 1980 11:36 AM)
-- Function: The implementation module for the Pilot OISCP Router's routing table.

DIRECTORY
  BufferDefs: FROM "BufferDefs", -- only for SHARES
  Checksums USING [IncrOisTransportControlAndUpdateChecksum],
  CommunicationInternal USING [], -- EXPORTS
  DriverDefs USING [
    doDebug, doStats, GetDeviceChain, GetSystemBufferPool, Glitch, Network],
  Heap USING [FreeNode, MakeNode],
  OISCPDefs USING [
    GetFreeOisBuffer, GetOisPacketTextLength, SetOisPacketTextLength,
    ReturnFreeOisBuffer, OisBuffer, OisNetID, OisHostID],
  OISCPTypes USING [
    allHostIDs, routingInformationSocket, unknownHostID, unknownNetID],
  Process USING [Detach, MsecToTicks, SetTimeout],
  Router USING [
    checkIt, FindMyHostID, routersFunction, Nibble, primaryMDS, RoutingTableEntry,
    RoutingTableObject, routingInfoRequest, routingInfoResponse, RoutingInfoTuple,
    SendPacket, XmitStatus],
  StatsDefs USING [StatIncr],
  SpecialCommunication USING [RoutersFunction],
  SpecialSystem USING [NetworkNumber];

RoutingTableImpl: MONITOR
  IMPORTS Checksums, DriverDefs, Heap, OISCPDefs, Process, Router, StatsDefs
  EXPORTS CommunicationInternal, Router, SpecialCommunication
  SHARES BufferDefs =
  BEGIN
  OPEN DriverDefs, OISCPDefs, OISCPTypes, Router, StatsDefs, SpecialCommunication;

  -- Many of these variables must eventually live in outerspace so that multiple MDSs
  -- access the same router variables. The modules in the primary MDS will have the
  -- proceses and will perform the initialization of the "globals", while the others will not.

  -- monitor data
  systemBufferPool: BufferDefs.BufferPool;
  myHostID: OISCPDefs.OisHostID; -- host ID of this system element
  -- routing table constants and variables
  -- network ids and now 32 bits. Therefore numberOfNets, LegalNets, destNet, net, will
  -- have to change. Rightt now we will use the low order bits of the Network Number;
  routingTableHead: RoutingTableEntry ← NIL;
  oneHop: Nibble = 1; -- delay to local network is assumd to be one router hop.
  updateCycles: Nibble = 4; -- number of routing table update cycles.
  maxRouterDelay: CARDINAL = 15; -- max number of router hops
  maxInternetrouterHops: CARDINAL = 15; -- max number of internetrouter hops
  -- controlling the entry processes
  iNRpleaseStop, pleaseStop: BOOLEAN; -- switch to tell the processes to stop
  routingTableUpdateTimer, responseFromInternetRouter, internetRouterTimer:
    CONDITION;
  routingTableFork, internetRouterServerFork: PROCESS;

  -- various Glitches generated by the Router
  RoutingTableScrambled: ERROR = CODE;

  --Hot Procedures

  -- Routing list manipulation Routines
```



```
  -- This procedure searches for an entry with destNetwork field equal to num and returns
  -- that entry and the previous entry. If the entry cannot be found then e is NIL and
  -- prev is undefined. If the entry is the first on the list then prev is NIL.
  FindNetworkNumber: PROCEDURE [num: SpecialSystem.NetworkNumber]
    RETURNS [e, prev: RoutingTableEntry] =
    BEGIN
    e ← routingTableHead;
    prev ← NIL;
    UNTIL (e = NIL) OR (e.destNetwork = num) DO prev ← e; e ← e.next; ENDLOOP;
    END;

  -- This procedure searches for an entry with network field equal to net and returns
  -- that entry and the previous entry. If the entry cannot be found then e is NIL and
  -- prev is undefined. If the entry is the first on the list then prev is NIL.

  FindNetwork: PROCEDURE [net: DriverDefs.Network]
    RETURNS [e, prev: RoutingTableEntry] =
    BEGIN
    e ← routingTableHead;
    prev ← NIL;
    UNTIL (e = NIL) OR (e.network = net) DO prev ← e; e ← e.next; ENDLOOP;
    END;

  -- This procedure removes RoutingTableEntry e from the list. prev is the previous entry
  -- to e. If prev is not known then its value is NIL.

  RemoveEntry: PROCEDURE [e: RoutingTableEntry, prev: RoutingTableEntry ← NIL] =
    BEGIN
    temp: RoutingTableEntry ← routingTableHead;
    IF prev = NIL THEN
      BEGIN
      UNTIL (temp = NIL) OR (temp = e) DO prev ← temp; temp ← temp.next; ENDLOOP;
      END;
    IF doDebug AND (temp = NIL) THEN Glitch[RoutingTableScrambled];
    IF prev = NIL THEN routingTableHead ← e.next
    ELSE
      BEGIN
      IF doDebug AND (prev.next # e) THEN Glitch[RoutingTableScrambled];
      IF doDebug AND (prev.next = NIL) THEN Glitch[RoutingTableScrambled];
      prev.next ← e.next;
      END;
    e.next ← NIL;
    END;

  -- This procedure adds an entry to the beginning of the Routingtable list.

  AddEntry: PROCEDURE [e: RoutingTableEntry] =
    BEGIN e.next ← routingTableHead; routingTableHead ← e; END;

  MoveToHeadOfTable: PROCEDURE [
    e: RoutingTableEntry, prev: RoutingTableEntry ← NIL] = INLINE
    BEGIN
    IF e # routingTableHead THEN BEGIN RemoveEntry[e, prev]; AddEntry[e]; END;
    END;


  FindNetworkAndTransmit: PUBLIC ENTRY PROCEDURE [b: OisBuffer]
    RETURNS [stat: XmitStatus] =
    BEGIN
```

```
    ENABLE UNWIND => NULL;
    destHost: OisHostID ← b.destination.host;
    nextHost: OisHostID;
    network: Network;
    e, prev: RoutingTableEntry;


      [e, prev] ← FindNetworkNumber[b.destination.net];
    IF e = NIL THEN
      BEGIN -- outgoing packet for unknown net,
      IF doStats THEN StatIncr[statOisSentNowhere];
      -- return b to the system buffer pool
      stat ← XmitStatus[noRouteToNetwork];
      IF b.pool = systemBufferPool THEN ReturnFreeOisBuffer[b]
      ELSE b.status ← LOOPHOLE[stat];
      RETURN;
      END;

    MoveToHeadOfTable[e, prev];
    -- outgoing packet to be transmitted over the correct network
    network ← e.network;
    IF (nextHost ← e.route) = unknownHostID THEN nextHost ← destHost; -- intranet
    stat ← XmitStatus[goodCompletion];
    network.encapsulateOis[b, nextHost];
    IF (b.pool # systemBufferPool) THEN
      BEGIN
      -- synchronous buffer send
      b.status ← LOOPHOLE[stat];
      b.sendInProgress ← TRUE;
      b.requeueProcedure ← NotifyTransmitCompleted;
      network.sendBuffer[b];
      WHILE (b.sendInProgress) DO WAIT b.sendCompleted; ENDLOOP;
      END
    ELSE network.sendBuffer[b]; -- system buffers are sent asynchronously
    RETURN;
    END; -- FindNetworkAndTransmit

  NotifyTransmitCompleted: ENTRY PROCEDURE [b: BufferDefs.Buffer] =
    BEGIN b.sendInProgress ← FALSE; NOTIFY b.sendCompleted; END;

  ForwardPacket: PUBLIC ENTRY PROCEDURE [b: OisBuffer] =
    BEGIN
    ENABLE UNWIND => NULL;
    nextHost: OisHostID;
    e, prev: RoutingTableEntry;
    network: Network;

    -- see if we have traversed max number of internet routers already
    IF b.oisTransportControl >= maxInternetrouterHops THEN
      BEGIN
      -- return b to the system buffer pool
      ReturnFreeOisBuffer[b];
      RETURN;
      END;


      [e, prev] ← FindNetworkNumber[b.destination.net];
    IF e = NIL THEN
```



```
      BEGIN -- outgoing packet for unknown net
      IF doStats THEN StatIncr[statOisNotForwarded];
      b.status ← LOOPHOLE[XmitStatus[noRouteToNetwork]];
      -- return b to the system buffer pool
      ReturnFreeOisBuffer[b];
      -- some day an error protocol packet will be sent back to the source

      END
    ELSE
      BEGIN -- outgoing packet
      MoveToHeadOfTable[e, prev];
      -- increment the hop count (oisTransportControl and update the checksum if appropriate
      IF checkIt THEN Checksums.IncrOisTransportControlAndUpdateChecksum[b]
      ELSE b.oisTransportControl ← b.oisTransportControl + 1;
      -- now transmit it over the correct network
      network ← e.network;
      IF (nextHost ← e.route) = unknownHostID THEN nextHost ← b.destination.host;
      -- same net
      network.encapsulateOis[b, nextHost];
      network.sendBuffer[b];
      IF doStats THEN StatIncr[statOisForwarded];
      END;
    END; -- ForwardPacket

  -- Cool Procedures
  -- This procedure tell the OISCP Router about a new network.

  AddNetwork: PUBLIC ENTRY PROCEDURE [newNetwork: Network] =
    BEGIN AddNetworkLocked[newNetwork]; END; -- AddNetwork

  -- This procedure tell the OISCP Router about a new network.

  AddNetworkLocked: PRIVATE PROCEDURE [newNetwork: Network] =
    BEGIN
    unknownNetIdEntry, e: RoutingTableEntry;
    IF NOT newNetwork.alive THEN RETURN;
    -- network must be alive to be added to table
    [e, ] ← FindNetworkNumber[newNetwork.netNumber];
    IF newNetwork.netNumber = unknownNetID THEN
      BEGIN
      Process.Detach[FORK ProbeAnInternetRouter[newNetwork]];
      -- this will help find the net number
      IF e # NIL THEN RETURN;
      -- do not add a network of unknown number if an unknown number network already exists

      END;
    IF e = NIL THEN
      BEGIN e ← Heap.MakeNode[n: SIZE[RoutingTableObject]]; AddEntry[e]; END;
    e↑ ← RoutingTableObject[
      next: e.next, destNetwork: newNetwork.netNumber, delay: oneHop,
      timeUnits: updateCycles, route: unknownHostID, network: newNetwork];
    -- if an unKnownNetID network does not exist, then then make an entry for it
    -- using this network.
    [unknownNetIdEntry, ] ← FindNetworkNumber[unknownNetID];
    IF unknownNetIdEntry = NIL THEN
      BEGIN
      unknownNetIdEntry ← Heap.MakeNode[n: SIZE[RoutingTableObject]];
      unknownNetIdEntry↑ ← RoutingTableObject[
        next:, destNetwork: unknownNetID, delay: oneHop, timeUnits: updateCycles,
```

```
      route: unknownHostID, network: newNetwork];
    AddEntry[unknownNetIdEntry];
    END;
  END; -- AddNetworkLocked

-- This procedure removes a network from the OISCP Router's tables.

RemoveNetwork: PUBLIC ENTRY PROCEDURE [oldNetwork: Network] =
  BEGIN
  RemoveNetworkLocked[oldNetwork];
  -- we may have removed the unknownNetID network; this will be replace by
  -- RoutingTableUpdater. It is not done here because of deadlock problems with Boss.
  NOTIFY routingTableUpdateTimer;
  END; -- RemoveNetwork

-- This procedure removes a network from the OISCP Router's tables.

RemoveNetworkLocked: PRIVATE PROCEDURE [oldNetwork: Network] =
  BEGIN
  e: RoutingTableEntry;
  DO
    [e, ] ← FindNetwork[oldNetwork];
    IF e = NIL THEN EXIT;
    RemoveEntry[e];
    Heap.FreeNode[p: e];
    ENDLOOP;
  END; -- RemoveNetworkLocked

StateChanged: PUBLIC ENTRY PROCEDURE [network: Network] =
  BEGIN
  RemoveNetworkLocked[network];
  AddNetworkLocked[network];
  -- if the state changed on the unknownNetID network then that entry was
  -- possibly deleted and NOT added back in; therefore, a new one must be found.
  -- The replacement is not done here because of deadlock problems with Boss.
  NOTIFY routingTableUpdateTimer;
  END; -- StateChanged

RoutingInformationPacket: PUBLIC PROCEDURE [b: OisBuffer] =
  BEGIN
  newDelay: INTEGER;
  objectNetID: OisNetID;
  sameRoute, betterDelay, awfulDelay, newNetwork: BOOLEAN;
  localRouterDelay: INTEGER = 1;
  newRoute: OisHostID = b.source.host;
  e: RoutingTableEntry;
  -- tricky, incomingNetwork must not become a dangling pointer while we don't have
  -- the monitor locked. This could happen if the network over which the packet arrived
  -- was removed. Sigh...
  incomingNetwork: Network = b.network;
  routingPacketType: CARDINAL = b.oisWords[0];
  routingInfoLength: CARDINAL = GetOisPacketTextLength[b] - 2; -- first word
  routingInfo: LONG POINTER TO RoutingInfoTuple ← LOOPHOLE[@b.oisWords[1]];

  UpdateUnnumberedNetTable: PROCEDURE = INLINE
    BEGIN
    IF incomingNetwork.netNumber # unknownNetID THEN RETURN;
    incomingNetwork.netNumber ← b.source.net;
    AddNetworkLocked[incomingNetwork];
```



```
    END; -- UpdateUnnumberedNetTable

  UpdateRoutingTable: PROCEDURE = INLINE
    BEGIN
    THROUGH [0..routingInfoLength/(2*SIZE[RoutingInfoTuple])) DO
      newDelay ← routingInfo.interrouterDelay + localRouterDelay;
      objectNetID ← routingInfo.objectNetID;
      routingInfo ← routingInfo + SIZE[RoutingInfoTuple];
      [e, ] ← FindNetworkNumber[objectNetID];
      IF e = NIL OR newDelay = oneHop THEN LOOP;
      sameRoute ← e.route = newRoute AND e.network = incomingNetwork;
      betterDelay ← newDelay < e.delay;
      awfulDelay ← newDelay > maxRouterDelay;
      newNetwork ← e.network = NIL;
      IF sameRoute THEN
        BEGIN
        e.delay ← newDelay;
        IF NOT awfulDelay THEN e.timeUnits ← updateCycles;
        END
      ELSE
        IF (newNetwork AND NOT awfulDelay) OR (NOT newNetwork AND betterDelay)
          THEN
          e↑ ← RoutingTableObject[
            next: e.next, destNetwork: objectNetID, delay: newDelay,
            timeUnits: updateCycles, route: newRoute, network: incomingNetwork];
      ENDLOOP;
    END; -- UpdateRoutingTable

  ExaminePacket: ENTRY PROCEDURE = INLINE
    BEGIN UpdateUnnumberedNetTable[]; UpdateRoutingTable[]; END;
  -- ExaminePacket

  -- main body of the procedure
  IF b.packetType # routingInformation OR newRoute = unknownHostID OR
    incomingNetwork = NIL THEN RETURN;
  IF doStats THEN StatIncr[statOisGatewayPacketsRecv];
  IF routersFunction = interNetworkRouting AND routingPacketType =
    routingInfoRequest THEN
    BEGIN
    -- we hope at this point that incomingNetwork isn't a dangling pointer
    SendRoutingInfoResponse[newRoute, incomingNetwork];
    RETURN;
    END;
  IF routingPacketType = routingInfoResponse THEN ExaminePacket[];
  END; -- RoutingInformationPacket

-- This procedure sends out Routing Information Protocol Request packets when
-- a new network is added to the OISCP Router's tables. If the Network were to
-- go away while we probe, we are in trouble, because we release the monitor when
-- we do a wait. In general this is a bad thing, but we asume that broadcast networks
-- like the Ethernet will be the only ones over which we can bind the network number
-- dynamically and they will be permenantish!

ProbeAnInternetRouter: PROCEDURE [network: Network] =
  BEGIN
  b: OisBuffer;

  ResponseFromInternetRouterPositive: ENTRY PROCEDURE RETURNS [BOOLEAN] = INLINE
    BEGIN
```

```
    WAIT responseFromInternetRouter;
    RETURN[network.netNumber # unknownNetID];
    END; -- ResponseFromInternetRouterPositive

  THROUGH [0..10) UNTIL pleaseStop DO
    b ← GetFreeOisBuffer[];
    b.oisTransportControl ← 0;
    b.packetType ← routingInformation;
    SetOisPacketTextLength[b, 2]; -- just one word of data
    b.source ← b.destination ←
      [net: unknownNetID, host: allHostIDs, socket: routingInformationSocket];
    b.source.host ← myHostID;
    b.oisWords[0] ← routingInfoRequest;
    Router.SendPacket[b];
    IF ResponseFromInternetRouterPositive[] THEN EXIT;
    ENDLOOP;
  END; -- ProbeAnInternetRouter

-- This procedure sends out Routing Information Protocol Response packet on request.
-- Note that this is not an ENTRY procedure even though we are touching a Network.
-- RouterSee had a pointer to the Network, and we hope that this hasn't become a
-- dangling pointer.

SendRoutingInfoResponse: PROCEDURE [to: OisHostID, network: Network] =
  BEGIN END; -- SendRoutingInfoResponse

-- This process wakes up every 30 seconds and sends out Routing Information
-- Protocol Response packets gratuitously, if this system element is an internet router.

InternetRouterServer: ENTRY PROCEDURE =
  BEGIN
  UNTIL pleaseStop OR iNRpleaseStop DO WAIT internetRouterTimer; ENDLOOP;
  END; -- InternetRouterServer

-- This process wakes up every 60 seconds. On awakening it goes through the
-- routing table entries decrementing by one the time since the entry was last
-- updated. If the time is zero, then the destination net is deemed unreachable.

RoutingTableUpdater: ENTRY PROCEDURE =
  BEGIN

  CheckUnknownNetIdEntry: PROCEDURE = INLINE
    BEGIN
    unknownNetIdEntry: RoutingTableEntry;
    network: DriverDefs.Network;
    [unknownNetIdEntry, ] ← FindNetworkNumber[unknownNetID];
    IF (unknownNetIdEntry = NIL) AND
      ((network ← DriverDefs.GetDeviceChain[]) # NIL) THEN
      AddNetworkLocked[network];
    END; -- CheckUnknownNetIdEntry

  rte: RoutingTableEntry;
  UNTIL pleaseStop DO
    -- no need to probe an internet router, since they broadcast every 30 secs.
    rte ← routingTableHead;
    WHILE (rte # NIL) DO
      IF rte.delay # oneHop THEN
        BEGIN
        IF rte.timeUnits = 0 THEN
```



```
          BEGIN
          temp: RoutingTableEntry ← rte;
          rte ← rte.next;
          RemoveEntry[temp];
          Heap.FreeNode[p: temp];
          LOOP;
          END
        ELSE rte.timeUnits ← rte.timeUnits - 1;
        END;
      rte ← rte.next;
      ENDLOOP;
    CheckUnknownNetIdEntry[];
    WAIT routingTableUpdateTimer;
    ENDLOOP;
  END; -- RoutingTableUpdater

-- This returns the ID of the locally connected networkmost suitable to get to the
-- destination network.

FindDestinationRelativeNetID: PUBLIC ENTRY PROCEDURE [destNet: OisNetID]
  RETURNS [OisNetID] =
  BEGIN
  e: RoutingTableEntry;
  [e] ← FindNetworkNumber[destNet];
  IF e = NIL THEN RETURN[unknownNetID] ELSE RETURN[e.network.netNumber];
  END; -- FindDestinationRelativeNetID

-- This procedure returns what the router is.

GetRouterFunction: PUBLIC ENTRY PROCEDURE RETURNS [RoutersFunction] =
  BEGIN RETURN[routersFunction]; END;

--Cold Procedures

-- The router must be told which function to perform.
-- Race condition with RoutingTableOn and RoutingTableOff concerning
-- FORKing and JOINing internetRouterServerFork. Races should not occur
-- because all of these are very Cold and only this procedure is exported
-- out of Pilot.

SetRouterFunction: PUBLIC PROCEDURE [newFunction: RoutersFunction]
  RETURNS [oldFunction: RoutersFunction] =
  BEGIN
  joinTheINRServer: BOOLEAN ← FALSE;
  zombieINRServer: PROCESS;

  SetFunction: ENTRY PROCEDURE = INLINE
    BEGIN
    oldFunction ← routersFunction;
    routersFunction ← newFunction;
    IF primaryMDS THEN
      SELECT TRUE FROM
        (newFunction = interNetworkRouting) AND (oldFunction = vanillaRouting)
          =>
          BEGIN
          iNRpleaseStop ← FALSE;
          internetRouterServerFork ← FORK InternetRouterServer[];
          END;
        (newFunction = vanillaRouting) AND (oldFunction = interNetworkRouting)
```

```
        =>
        BEGIN
        joinTheINRServer ← iNRpleaseStop ← TRUE;
        zombieINRServer ← internetRouterServerFork;
        NOTIFY internetRouterTimer;
        END;
      ENDCASE => NULL;
  END; -- inline of SetFunction

-- mainline code of SetRouterFunction
SetFunction[];
IF joinTheINRServer THEN JOIN zombieINRServer;
END;


-- This procedure turns the router on. The Communication software is written with the
-- idea of eventually turning the code on and off during execution, and so we should
-- get the latest values of netID when being turned back on. The processes asociated
-- with the routing table should be created only if this module is in the primary MDS.

RoutingTableOn: PUBLIC PROCEDURE =
  BEGIN
  iNRpleaseStop ← pleaseStop ← FALSE;
  IF primaryMDS THEN
    BEGIN
    network: DriverDefs.Network ← DriverDefs.GetDeviceChain[];
    RoutingTableActivate[];
    FOR network ← network, network.next UNTIL network = NIL DO
      AddNetwork[network]; ENDLOOP;
    IF routersFunction = interNetworkRouting THEN
      internetRouterServerFork ← FORK InternetRouterServer[];
    routingTableFork ← FORK RoutingTableUpdater[];
    END;
  END; -- RoutingTableOn

RoutingTableActivate: ENTRY PROCEDURE = INLINE
  BEGIN
  routingTableHead ← NIL;
  systemBufferPool ← DriverDefs.GetSystemBufferPool[];
  myHostID ← Router.FindMyHostID[];
  END; -- RoutingTableActivate

RoutingTableOff: PUBLIC PROCEDURE =
  BEGIN
  IF primaryMDS THEN
    BEGIN
    RoutingTableDeactivate[];
    JOIN routingTableFork[];
    IF routersFunction = interNetworkRouting THEN JOIN internetRouterServerFork;
    CleanUpRoutingTable[];
    END;
  END; -- RoutingTableOff

RoutingTableDeactivate: ENTRY PROCEDURE = INLINE
  BEGIN
  iNRpleaseStop ← pleaseStop ← TRUE;
  NOTIFY routingTableUpdateTimer;
  BROADCAST responseFromInternetRouter;
  IF routersFunction = interNetworkRouting THEN NOTIFY internetRouterTimer;
```



```
  END; -- RoutingTableDeactivate

CleanUpRoutingTable: ENTRY PROCEDURE = INLINE
  BEGIN
  e, temp: RoutingTableEntry;
  e ← routingTableHead;
  WHILE e # NIL DO temp ← e; e ← e.next; Heap.FreeNode[p: temp]; ENDLOOP;
  END; -- RoutingTableDeactivate

-- initialization (Cold)

Process.SetTimeout[@routingTableUpdateTimer, Process.MsecToTicks[60000]];
Process.SetTimeout[@responseFromInternetRouter, Process.MsecToTicks[1000]];
Process.SetTimeout[@internetRouterTimer, Process.MsecToTicks[30000]];

END. -- RoutingTableImpl module.
```

LOG

Time: January 19, 1980 4:05 PM By: Dalal Action: Split OISCPRouter into two.
Time: March 13, 1980 5:11 PM By: BLyon Action: FindNetworkAndTransmit synchronously sends
***non-system buffers and asynchronously sends system buffers.*
Time: July 31, 1980 11:53 AM By: BLyon Action: replaced internetRouter with routersFunction an
***d added Get/SetRoutersFunction.*
Time: August 5, 1980 4:25 PM By: BLyon Action: RoutingTable is now linked list and no such con
***cept as primaryNetID.*
Time: September 13, 1980 6:15 PM By: HGM Action: Add StateChanged.
Time: September 18, 1980 3:16 PM By: BLyon Action: modified StateChanged, Add/Delete Netw
***ork .., removed FindPrimaryNetID.*

```
-- PilotCommUtil.mesa (last edited by: BLyon on: August 11, 1980 4:52 PM)
-- Function: The implementation module for Pilot provided utilities.

DIRECTORY
  CommunicationInternal USING [],
  CommUtilDefs USING [],
  DriverDefs USING [],
  Environment USING [wordsPerPage],
  Inline USING [COPY, LongCOPY, LowHalf],
  Process USING [MsecToTicks, SecondsToTicks, Ticks],
  PupDefs USING [Tocks], --EXPORTS Time Conversion stuff
  ResidentHeap USING [MakeNode, FreeNode, first64K],
  Runtime USING [CallDebugger],
  Space USING [
    Handle, Create, defaultBase, defaultWindow, GetAttributes, GetHandle,
    PageFromLongPointer, Map, PageCount, Unmap, Delete, virtualMemory,
    LongPointer],
  SpecialSpace USING [MakeSwappable, MakeResident],
  Zone USING [Status, BlockSize, Base];

PilotCommUtil: PROGRAM
  IMPORTS Inline, Process, ResidentHeap, Runtime, Space, SpecialSpace
  EXPORTS CommunicationInternal, CommUtilDefs, DriverDefs, PupDefs =
  BEGIN

  -- Heap nodes.  All these are currently resident, and in first 64K
  ResidentZoneTrouble: PUBLIC ERROR = CODE;

  -- IOCB allocation.
  AllocateIocbs: PUBLIC PROC [nwords: CARDINAL] RETURNS [LONG POINTER] =
    BEGIN
    status: Zone.Status;
    p: Zone.Base RELATIVE POINTER;
    [p, status] ← ResidentHeap.MakeNode[LOOPHOLE[nwords, Zone.BlockSize], a4];
    IF status # okay THEN Glitch[ResidentZoneTrouble];
    RETURN[@ResidentHeap.first64K[p]]; -- make into a LONG absolute

    END;

  FreeIocbs: PUBLIC PROC [iocbPtr: LONG POINTER] =
    BEGIN
    p: Zone.Base RELATIVE POINTER ← Inline.LowHalf[
      iocbPtr - LOOPHOLE[ResidentHeap.first64K, LONG POINTER]];
    [] ← ResidentHeap.FreeNode[p];
    END;

  -- Clumps of pages

  AllocateBuffers: PUBLIC PROC [nwords: CARDINAL] RETURNS [base: LONG POINTER] =
    BEGIN
    pages: CARDINAL =
      (nwords + Environment.wordsPerPage - 1)/Environment.wordsPerPage;
    h: Space.Handle = Space.Create[pages, Space.virtualMemory, Space.defaultBase];
    Space.Map[h, Space.defaultWindow];
    base ← Space.LongPointer[h];
    END;

  FreeBuffers: PUBLIC PROC [base: LONG POINTER] =
```



```
    BEGIN
    pages: Space.PageCount;
    h: Space.Handle = Space.GetHandle[Space.PageFromLongPointer[base]];
    [, , , , pages, ] ← Space.GetAttributes[h];
    Space.Unmap[h];
    Space.Delete[h];
    END;

  LockBuffers: PUBLIC PROC [p: LONG POINTER] =
    BEGIN
    SpecialSpace.MakeResident[Space.GetHandle[Space.PageFromLongPointer[p]]];
    END;

  UnlockBuffers: PUBLIC PROC [p: LONG POINTER] =
    BEGIN
    SpecialSpace.MakeSwappable[Space.GetHandle[Space.PageFromLongPointer[p]]];
    END;


  -- Other goodies

  Zero: PUBLIC PROC [p: POINTER, l: CARDINAL] =
    BEGIN
    IF l = 0 THEN RETURN;
    p↑ ← 0;
    Inline.COPY[from: p, to: p + 1, nwords: l - 1];
    END;

  -- TimeCoversion for Pup package
  -- implement with Process.Tick = Tock
  -- Code depends upon Process.MsecToTicks corectly
  -- rounding up the argument before caculating the result,
  -- and SecondsToTicks returning LAST[Ticks] for large seconds.

  TockConversionTroubles: PUBLIC ERROR = CODE;

  SecondsToTocks: PUBLIC PROC [t: CARDINAL] RETURNS [PupDefs.Tocks] =
    BEGIN
    tks: Process.Ticks = Process.SecondsToTicks[t];
    IF tks = LAST[Process.Ticks] THEN ERROR TockConversionTroubles;
    RETURN[LOOPHOLE[tks]];
    END;

  -- this assumes that a tock is longer than an millisecond

  MsToTocks: PUBLIC PROC [t: CARDINAL] RETURNS [PupDefs.Tocks] =
    BEGIN
    IF t = 0 THEN ERROR TockConversionTroubles;
    RETURN[LOOPHOLE[Process.MsecToTicks[t]]];
    END;

  -- Communication error handler

  errorHandler: PROC [ERROR] ← DefaultErrorHandler;

  CaptureErrors: PUBLIC PROC [proc: PROC [ERROR]] = {errorHandler ← proc};

  DefaultErrorHandler: PROC [why: ERROR] =
    BEGIN Runtime.CallDebugger["CommunicationGlitch"L]; Glitch[why]; END;
```

```
  Glitch: PUBLIC PROC [why: ERROR] = {errorHandler[why]};

  CopyLong: PUBLIC PROC [from: LONG POINTER, nwords: CARDINAL, to: LONG POINTER] =
    {Inline.LongCOPY[from, nwords, to]};

  END.... -- PilotCommUtil module

LOG

Time: June 18, 1979  2:13 PM  By: Dalal  Action: conversion to Pilot 3.0.
Time: August 25, 1979  1:40 PM  By: Dalal  Action: made msPerTick a constant.
Time: August 31, 1979  12:34 PM  By: Dalal  Action: deleted AllocateHeapNode.
Time: August 11, 1980  4:53 PM  By: BLyon  Action: deleted ByteBlt*.
-- things needed by the device drivers
InterruptLevel: TYPE = ProcessInternal.NakedNotifyLevel;

AddInterruptHandler: PUBLIC PROC [
   i: InterruptLevel, c: POINTER TO CONDITION, w: WORD] =
  BEGIN
  ProcessInternal.SetNakedNotifyPointer[i, c];
  ProcessInternal.DisableInterrupts[];
  SDDefs.SD[SDDefs.slOResetBits] ←
    Inline.BITOR[SDDefs.SD[SDDefs.slOResetBits], w];
  ProcessInternal.EnableInterrupts[];
  END;

RemoveInterruptHandler: PUBLIC PROC [i: InterruptLevel] =
  BEGIN
  ProcessInternal.SetNakedNotifyPointer[i, NIL];
  END;
```

```
-- QueueImpl.mesa (last edited by: HGM/BLyon on: May 31, 1980 3:08 PM)

DIRECTORY
  BufferDefs USING [
    Buffer, OisBuffer, SppBuffer, PupBuffer, RppBuffer, Queue, BufferPool,
    ReturnFreeBuffer, ReturnBufferToPool],
  DriverDefs USING [doDebug, doStats, Glitch, GetSystemBufferPool],
  DriverTypes USING [bufferSeal, queueSeal],
  OISCPDefs USING [],
  PupDefs USING [],
  StatsDefs USING [StatIncr];

QueueImpl: PROGRAM
  IMPORTS BufferDefs, DriverDefs, StatsDefs
  EXPORTS BufferDefs, DriverDefs, OISCPDefs, PupDefs
  SHARES BufferDefs =
  BEGIN OPEN BufferDefs, DriverDefs;

  -- NB: Caller must protect his own Queues.

  QueueScrambled: PUBLIC ERROR = CODE;
  QueueSealBroken: PUBLIC ERROR = CODE;
  BufferSealBroken: PUBLIC ERROR = CODE;

  -- Cool procedures

  QueueInitialize: PUBLIC PROCEDURE [q: Queue] =
    BEGIN
    q↑ ← [length: 0, first: NIL, last: NIL, seal: DriverTypes.queueSeal];
    END;

  -- Put all buffers back onto buffer.pool's freeQueue

  QueueCleanup: PUBLIC PROCEDURE [q: Queue] =
    BEGIN
    sbp: BufferPool = DriverDefs.GetSystemBufferPool[];
    b: Buffer;
    UNTIL (b ← Dequeue[q]) = NIL DO
      IF b.pool = sbp THEN ReturnFreeBuffer[b] ELSE ReturnBufferToPool[b.pool, b];
      ENDLOOP;
    END;

  ExtractPupFromQueue: PUBLIC PROCEDURE [Queue, PupBuffer] RETURNS [PupBuffer] =
    LOOPHOLE[ExtractFromQueue];
  ExtractRppFromQueue: PUBLIC PROCEDURE [Queue, RppBuffer] RETURNS [RppBuffer] =
    LOOPHOLE[ExtractFromQueue];
  ExtractOisFromQueue: PUBLIC PROCEDURE [Queue, OisBuffer] RETURNS [OisBuffer] =
    LOOPHOLE[ExtractFromQueue];
  ExtractSppFromQueue: PUBLIC PROCEDURE [Queue, SppBuffer] RETURNS [SppBuffer] =
    LOOPHOLE[ExtractFromQueue];
  ExtractFromQueue: PUBLIC PROCEDURE [q: Queue, b: Buffer] RETURNS [Buffer] =
    BEGIN
    previousB, currentB: Buffer;
    IF q = NIL THEN Glitch[QueueScrambled];
    IF doDebug AND q.seal # DriverTypes.queueSeal THEN Glitch[QueueSealBroken];
    IF doDebug AND b.seal # DriverTypes.bufferSeal THEN Glitch[BufferSealBroken];
    previousB ← NIL;
    currentB ← q.first;
```



```
    UNTIL currentB = b DO
      IF currentB = NIL THEN EXIT;
      previousB ← currentB;
      currentB ← currentB.next;
      ENDLOOP;
    IF currentB # NIL THEN
      BEGIN
      -- remove this buffer from the queue
      IF doDebug AND currentB.seal # DriverTypes.bufferSeal THEN
        Glitch[BufferSealBroken];
      IF currentB = q.first THEN q.first ← currentB.next;
      IF currentB = q.last THEN q.last ← previousB;
      IF previousB # NIL THEN previousB.next ← currentB.next;
      q.length ← q.length - 1;
      currentB.queue ← NIL;
      currentB.next ← NIL;
      IF doStats THEN StatsDefs.StatIncr[statXqueue];
      END
    ELSE IF doStats THEN StatsDefs.StatIncr[statXqueueNIL];
    RETURN[currentB];
    END;

  -- Hot procedures

  EnqueuePup: PUBLIC PROCEDURE [Queue, PupBuffer] = LOOPHOLE[Enqueue];
  EnqueueRpp: PUBLIC PROCEDURE [Queue, RppBuffer] = LOOPHOLE[Enqueue];
  EnqueueOis: PUBLIC PROCEDURE [Queue, OisBuffer] = LOOPHOLE[Enqueue];
  EnqueueSpp: PUBLIC PROCEDURE [Queue, SppBuffer] = LOOPHOLE[Enqueue];
  Enqueue: PUBLIC PROCEDURE [q: Queue, b: Buffer] =
    BEGIN
    IF q = NIL OR b = NIL OR b.queue # NIL THEN Glitch[QueueScrambled];
    IF doDebug AND q.seal # DriverTypes.queueSeal THEN Glitch[QueueSealBroken];
    IF doDebug AND b.seal # DriverTypes.bufferSeal THEN Glitch[BufferSealBroken];
    IF doDebug AND q.length # 0 AND (q.first = NIL OR q.last = NIL) THEN
      Glitch[QueueScrambled];
    IF doDebug AND q.length > 256 THEN Glitch[QueueScrambled];
    b.next ← NIL;
    IF doStats THEN StatsDefs.StatIncr[statEnqueue];
    IF q.first = NIL THEN q.first ← b ELSE q.last↑.next ← b;
    q.last ← b;
    b.queue ← q;
    q.length ← q.length + 1;
    END; -- Enqueue

  DequeuePup: PUBLIC PROCEDURE [Queue] RETURNS [PupBuffer] = LOOPHOLE[Dequeue];
  DequeueRpp: PUBLIC PROCEDURE [Queue] RETURNS [RppBuffer] = LOOPHOLE[Dequeue];
  DequeueOis: PUBLIC PROCEDURE [Queue] RETURNS [OisBuffer] = LOOPHOLE[Dequeue];
  DequeueSpp: PUBLIC PROCEDURE [Queue] RETURNS [SppBuffer] = LOOPHOLE[Dequeue];
  Dequeue: PUBLIC PROCEDURE [q: Queue] RETURNS [b: Buffer] =
    BEGIN
    IF q = NIL THEN Glitch[QueueScrambled];
    IF doDebug AND q.seal # DriverTypes.queueSeal THEN Glitch[QueueSealBroken];
    IF (b ← q.first) = NIL THEN
      BEGIN
      IF doDebug AND q.length # 0 THEN Glitch[QueueScrambled];
      IF doStats THEN StatsDefs.StatIncr[statDequeueNIL];
      RETURN;
      END;
    IF (q.first ← q.first.next) = NIL THEN q.last ← NIL;
```

```
    IF doDebug AND q.length > 256 THEN Glitch[QueueScrambled];
    q.length ← q.length - 1;
    IF doStats THEN StatsDefs.StatIncr[statDequeue];
    IF b.queue # q THEN Glitch[QueueScrambled];
    b.queue ← NIL;
    b.next ← NIL;
    IF doDebug AND b.seal # DriverTypes.bufferSeal THEN Glitch[BufferSealBroken];
    IF doDebug AND q.length # 0 AND (q.first = NIL OR q.last = NIL) THEN
      Glitch[QueueScrambled];
    END; -- Dequeue

  -- initialization

  END. -- QueueImpl module
```

LOG

Time: April 17, 1980 5:06 PM By: Dalal Action: created file for Pilot 5.0.

```
-- SocketImpl.mesa (last edited by: BLyon on: October 9, 1980 9:54 AM)
-- Function: The implementation module for Pilot OISCP Socket Channels.

DIRECTORY
  BufferDefs USING [
    BufferPool, BufferPoolObject, BufferPoolMake, BufferPoolDestroy,
    QueueInitialize, QueueCleanup],
  CommunicationInternal USING [],
  CommUtilDefs USING [CopyLong, MaybeShorten],
  Heap USING [MakeNode, FreeNode],
  Inline USING [LowHalf],
  OISCPDefs USING [
    DequeueOis, EnqueueOis, ExtractOisFromQueue, OisAddress, OisBuffer, OisHostID,
    OisNetID, QueueObject, GetFreeSendOisBufferFromPool,
    GetFreeReceiveOisBufferFromPool, ReturnSendOisBufferToPool,
    ReturnReceiveOisBufferToPool],
  OISCPTypes USING [
    bytesPerOisPktText, maxDataWordsPerSpp, uniqueAddress, unknownHostID,
    unknownNetID, unknownSocketID],
  Process USING [
    InitializeCondition, MsecToTicks, SecondsToTicks, SetTimeout, Ticks, Yield],
  PupTypes USING [maxDataWordsPerGatewayPup],
  Router USING [
    AddSocket, AssignOisAddress, FindDestinationRelativeNetID, FindMyHostID,
    RemoveSocket, socketRouterLock, SendPacket, XmitStatus],
  Socket USING [
    defaultWaitTime, PhysicalRecordHandle, OISPktHeaderRecord, SocketStatus,
    TransferStatus, WaitTime],
  SocketInternal USING [BufferPoolType, SocketHandle, SocketObject],
  SpecialSystem USING [NetworkAddress, nullNetworkAddress],
  System USING [GetClockPulses, Pulses, NetworkAddress, MicrosecondsToPulses];

SocketImpl: MONITOR LOCKS Router.socketRouterLock
  IMPORTS
    BufferDefs, CommUtilDefs, Heap, Inline, OISCPDefs, Process, Router, System
  EXPORTS
    CommunicationInternal, Socket, SocketInternal,
    -- others --System
  SHARES BufferDefs =
  BEGIN OPEN OISCPDefs, OISCPTypes, Socket, SocketInternal;

  -- EXPORTED TYPES and READONLY Variables
  ChannelHandle: PUBLIC TYPE = SocketInternal.SocketHandle;
  GetHandle: PUBLIC TYPE = OISCPDefs.OisBuffer;
  NetworkAddress: PUBLIC TYPE = SpecialSystem.NetworkAddress; -- others
  uniqueNetworkAddr: PUBLIC System.NetworkAddress ←
    SpecialSystem.nullNetworkAddress;

  -- All socket are covered by one lock. We must be careful when we go to multiple
  -- MDSs as we must make sure to specify where the lock really is, i.e. in outerspace.

  -- constants for various kinds of buffer pools
  dataWordsPerPacketStreamBuffer: CARDINAL = MAX[
    PupTypes.maxDataWordsPerGatewayPup, OISCPTypes.maxDataWordsPerSpp];
  crateSend: CARDINAL = 3;
  crateReceive: CARDINAL = 3;
  dataWordsPerCrateBuffer: CARDINAL = 0;
```



```
  -- lets packaging get correct buffers
  BufferOperation: TYPE = {sending, receiving};

  -- local copies for speed
  myHostID: OISCPDefs.OisHostID;

  -- Signals and Errors
  TimeOut: PUBLIC ERROR = CODE;
  ChannelAborted: PUBLIC ERROR = CODE;
  ChannelError: PUBLIC ERROR = CODE;

  -- Hot Procedures

  GetPulsesIntervalTime: PROCEDURE [startTime: System.Pulses]
    RETURNS [LONG CARDINAL] = INLINE
    BEGIN
    RETURN[LOOPHOLE[System.GetClockPulses[] - startTime, LONG CARDINAL]];
    END; -- end Put Procedure

  -- This procedure packages a record into a crate and send the resulting packet out from
  -- the socket channel.

  Put: PUBLIC PROCEDURE [cH: ChannelHandle, recH: PhysicalRecordHandle]
    RETURNS [TransferStatus] =
    BEGIN
    b: OISCPDefs.OisBuffer;
    stat: Socket.TransferStatus;
    b ← GetFreeSendOisBufferFromPool[@cH.poolObject];
    PackageIntoCrate[recH, b];
    PutPacket[cH, b];
    stat ← LOOPHOLE[b.status, Socket.TransferStatus];
    ReturnSendOisBufferToPool[b.pool, b];
    RETURN[stat];
    END; -- end Put Procedure

  -- This procedure puts the buffer containing a packet out from this socket channel.

  PutPacket: PUBLIC PROCEDURE [cH: ChannelHandle, b: OISCPDefs.OisBuffer] =
    BEGIN

    PutLocked: ENTRY PROCEDURE = INLINE
      BEGIN
      IF cH.channelState # active THEN RETURN WITH ERROR ChannelAborted;
      b.status ← LOOPHOLE[Router.XmitStatus[goodCompletion]];
      -- assume all will go OK

      END; -- PutLocked

    -- fix up the buffer
    PutLocked[];
    b.oisTransportControl ← 0;
    b.source ← cH.localAddr;
    IF b.source.net = unknownNetID THEN
      b.source.net ← Router.FindDestinationRelativeNetID[b.destination.net];
    IF b.destination.host = unknownHostID OR b.destination.socket =
      unknownSocketID THEN
      BEGIN
      b.status ← LOOPHOLE[Router.XmitStatus[invalidDestAddr]]; -- this is safe!
      RETURN;
```

```
    END;
  Router.SendPacket[b];
  -- the following Yield is to prevent local communication from using all of the cycles.
  IF b.destination.host = myHostID THEN Process.Yield[];
  END; -- PutPacket

-- This procedure supplies a crate into which the client wants the next packet that
-- arrives on this socket channel.

Get: PUBLIC ENTRY PROCEDURE [cH: ChannelHandle, recH: PhysicalRecordHandle]
  RETURNS [b: GetHandle] =
  BEGIN
  IF recH = NIL THEN RETURN WITH ERROR ChannelError;
  b ← GetFreeReceiveOisBufferFromPool[@cH.poolObject];
  PackageIntoCrate[recH, b];
  IF cH.channelState # active THEN RETURN WITH ERROR ChannelAborted;
  b.status ← LOOPHOLE[Router.XmitStatus[pending]];
  EnqueueOis[@cH.pendingUserGetQueue, b];
  END; -- Get

-- Gives the clock time in milliseconds
-- This procedure waits for the completion of an operation by a previous Get call.

TransferWait: PUBLIC ENTRY PROCEDURE [cH: ChannelHandle, getH: GetHandle]
  RETURNS [recH: PhysicalRecordHandle, status: TransferStatus] =
  BEGIN
  startTime: System.Pulses ← System.GetClockPulses[];
  DO
    IF LOOPHOLE[getH.status, Router.XmitStatus] # pending THEN
      BEGIN
      [] ← ExtractOisFromQueue[@cH.completedUserGetQueue, getH];
      recH ← LOOPHOLE[ShortenPointer[getH.userPtr], PhysicalRecordHandle];
      status ← LOOPHOLE[getH.status, TransferStatus];
      ReturnReceiveOisBufferToPool[@cH.poolObject, getH];
      EXIT;
      END
    ELSE
      BEGIN
      IF (cH.channelState = aborted) THEN RETURN WITH ERROR ChannelAborted;
      IF GetPulsesIntervalTime[startTime] > cH.waitTime THEN
        RETURN WITH ERROR TimeOut;
      WAIT cH.newUserInput;
      END;
    ENDLOOP;
  END; -- TransferWait

-- This procedure waits for the completion of an operation by a previous Get call.

TransferWaitAny: PUBLIC ENTRY PROCEDURE [sH: ChannelHandle]
  RETURNS [recH: PhysicalRecordHandle, status: TransferStatus] =
  BEGIN
  b: OISCPDefs.OisBuffer;
  startTime: System.Pulses ← System.GetClockPulses[];
  DO
    IF (b ← DequeueOis[@sH.completedUserGetQueue]) # NIL THEN
      BEGIN
      recH ← LOOPHOLE[ShortenPointer[b.userPtr], PhysicalRecordHandle];
      status ← LOOPHOLE[b.status, TransferStatus];
      ReturnReceiveOisBufferToPool[@sH.poolObject, b];
      EXIT;
```



```
      END
    ELSE
      BEGIN
      IF (sH.channelState = aborted) THEN RETURN WITH ERROR ChannelAborted;
      IF GetPulsesIntervalTime[startTime] > sH.waitTime THEN
        RETURN WITH ERROR TimeOut;
      WAIT sH.newUserInput;
      END;
    ENDLOOP;
  END; -- TransferWaitAny

-- This procedure blocks until a buffer appears the socket's completedReceiveQueue.
-- The buffer is obtained by the dispatcher for the socket's freeQueue

GetPacket: PUBLIC ENTRY PROCEDURE [sH: SocketHandle] RETURNS [b: OisBuffer] =
  BEGIN
  startTime: System.Pulses ← System.GetClockPulses[];
  WHILE (b ← DequeueOis[@sH.completedReceiveQueue]) = NIL DO
    IF (sH.channelState = aborted) THEN RETURN WITH ERROR ChannelAborted;
    IF GetPulsesIntervalTime[startTime] > sH.waitTime THEN
      RETURN WITH ERROR TimeOut;
    WAIT sH.newReceiveInput;
    ENDLOOP;
  END;

-- This procedure packages a physical record into a crate buffer.

PackageIntoCrate: PRIVATE PROCEDURE [
  recH: PhysicalRecordHandle, b: OISCPDefs.OisBuffer] =
  BEGIN
  dataLen: CARDINAL ← recH.header.pktLength - 2*SIZE[Socket.OISPktHeaderRecord];
  -- in bytes
  -- We assume that the start of the block is word alligned and packet is right size.
  IF (dataLen > bytesPerOisPktText) THEN ERROR ChannelError;
  -- copy the header
  CommUtilDefs.CopyLong[
    from: @recH.header, nwords: SIZE[OISPktHeaderRecord], to: @b.checksum];
  -- hang the header and data pointer from appropriate pointers in b.
  b.userPtr ← recH;
  -- this now becomes a LONG, so shorten during unpackage operation
  b.userData ← @recH.body;
  b.userDataLength ← dataLen; -- in bytes
  RETURN;
  END; -- PackageIntoCrate

ShortenPointer: PROCEDURE [long: LONG POINTER] RETURNS [POINTER] = INLINE
  BEGIN RETURN[Inline.LowHalf[long]]; END; -- ShortenPointer

--Cool Procedures

-- This procedure assigns a temporary socket number in a NetworkAddress.

AssignNetworkAddress: PUBLIC PROCEDURE RETURNS [System.NetworkAddress] =
  BEGIN RETURN[Router.AssignOisAddress[]]; END; -- AssignNetworkAddress

-- This procedure makes a socket channel with the specified socket number for
-- the average client, by specifying that the socket has crates that hold client blocks.

Create: PUBLIC PROCEDURE [local: NetworkAddress] RETURNS [ChannelHandle] =
```

```
  BEGIN RETURN[CreateInternal[local, crate, crateSend, crateReceive]]; END;
-- Create

-- This procedure deletes this socket channel by deleting the socket from the router's
-- tables, and cleans up the data structures associated with this socket.

Delete: PUBLIC PROCEDURE [sH: ChannelHandle] =
  BEGIN

  DeleteLocked: ENTRY PROCEDURE = INLINE
    BEGIN
    ENABLE UNWIND => NULL;
    BufferDefs.QueueCleanup[@sH.completedReceiveQueue];
    BufferDefs.QueueCleanup[@sH.completedUserGetQueue];
    BufferDefs.QueueCleanup[@sH.pendingUserGetQueue];
    BufferDefs.BufferPoolDestroy[@sH.poolObject];
    END; -- DeleteLocked

  -- This procedure is a bit tricky to avoid any monitor problems with socket handles.
  -- We want to make sure that the process that from the
  -- router always has a valid socket handle, and that there are no deadlocks.
  Router.RemoveSocket[sH];
  DeleteLocked[];
  Heap.FreeNode[p: sH];
  END; --

-- This procedure aborts I/O on this socket channel.

Abort: PUBLIC ENTRY PROCEDURE [sH: ChannelHandle] =
  BEGIN
  b: OisBuffer;
  sH.channelState ← aborted;
  -- We allow all puts to complete on their own since they will reasonably soon!
  -- For each OisBuffer on the pendingGetQueue make its status = aborted
  UNTIL (b ← DequeueOis[@sH.pendingUserGetQueue]) = NIL DO
    b.status ← LOOPHOLE[Router.XmitStatus[aborted]];
    EnqueueOis[@sH.completedUserGetQueue, b];
    ENDLOOP;
  BROADCAST sH.newUserInput; -- tell all potential waiters
  BROADCAST sH.newReceiveInput; -- tell all potential waiters

  END; -- Abort

Reset: PUBLIC ENTRY PROCEDURE [sH: ChannelHandle] =
  BEGIN sH.channelState ← active; END;

-- This procedure gets the status of this socket channel.

GetStatus: PUBLIC ENTRY PROCEDURE [sH: ChannelHandle]
  RETURNS [status: SocketStatus] =
  BEGIN
  status ←
    [localAddr: sH.localAddr, state: sH.channelState,
      incompleteGets: sH.pendingUserGetQueue.length];
  END; -- GetStatus

-- This procedure sets the wait time for this socket channel.

SetWaitTime: PUBLIC ENTRY PROCEDURE [cH: ChannelHandle, time: WaitTime] =
```



```
  BEGIN
  ticks: Process.Ticks;
  condTime: CARDINAL ← Inline.LowHalf[time];
  lastCard: LONG CARDINAL = LAST[CARDINAL]; -- no, not last LONG CARDINAL !!
  cH.waitTime ← MilliSecondsToPulses[time];
  -- fix the condition variables to reflect this change
  -- this is a bit hairy because ticks are (short) CARDINALs
  IF time > lastCard THEN
    BEGIN
    timeInSeconds: LONG CARDINAL ← time/1000;
    IF timeInSeconds > lastCard THEN condTime ← LAST[CARDINAL] - 1
      -- still too big !!!

    ELSE condTime ← Inline.LowHalf[timeInSeconds];
    ticks ← Process.SecondsToTicks[condTime];
    END
  ELSE ticks ← Process.MsecToTicks[condTime] + 1;
  Process.SetTimeout[CommUtilDefs.MaybeShorten[@cH.newUserInput], ticks];
  Process.SetTimeout[CommUtilDefs.MaybeShorten[@cH.newReceiveInput], ticks];
  END; -- SetWaitTime

pulsesPerMilliSecond: LONG CARDINAL;
MilliSecondsToPulses: PROCEDURE [ms: LONG CARDINAL]
  RETURNS [pulses: LONG CARDINAL] =
  BEGIN
  -- we must be carefull about multiplication overflow since milliSeconds must be
  -- converted to microSeconds
  IF ms >= LAST[LONG CARDINAL]/1000 -- overflow condition
    THEN
    IF ms >= LAST[LONG CARDINAL]/pulsesPerMilliSecond -- ms is out of range
      THEN pulses ← LAST[LONG CARDINAL]
    ELSE pulses ← ms*pulsesPerMilliSecond -- close guestimation

  ELSE pulses ← LOOPHOLE[System.MicrosecondsToPulses[1000*ms], LONG CARDINAL];
  END; -- end MilliSecondsToPulses


-- This procedure makes a socket channel with the specified socket number,
-- and makes the right kind of buffer pool for the socket. If the value of local is
-- uniqueAddress then an unused address is assigned to the socket.
-- Some day we will check to see that the socket number requested is valid.

CreateInternal: PUBLIC PROCEDURE [
  local: OISCPDefs.OisAddress, bufferPoolType: BufferPoolType,
  send, receive: CARDINAL] RETURNS [sH: SocketHandle] =
  BEGIN
  sH ← LOOPHOLE[Heap.MakeNode[n: SIZE[SocketObject]]];
  -- fill in all the specific fields of the socket object appropriately,
  -- make the appropriate buffer pool, which can be the the spp buffer pool per socket,
  -- the listener buffer pool per socket, or just a pool of small buffers called crates.
  sH.poolObject.send ← send;
  sH.poolObject.receive ← receive;
  sH.poolObject.total ← send + receive;
  SELECT (sH.bufferPoolType ← bufferPoolType) FROM
    normal =>
      BEGIN
      sH.poolObject.dataWordsPerBuffer ← dataWordsPerPacketStreamBuffer;
      sH.poolObject.bufferSize ← big;
      -- all other fields initialized by BufferPoolMake
```

```
      END;
    crate =>
      BEGIN
      sH.poolObject.dataWordsPerBuffer ← dataWordsPerCrateBuffer;
      sH.poolObject.bufferSize ← user;
      -- all other fields initialized by BufferPoolMake

      END;
    ENDCASE => ERROR;
  BufferDefs.BufferPoolMake[@sH.poolObject];
  sH.localAddr ←
    IF local = OISCPTypes.uniqueAddress THEN Router.AssignOisAddress[]
    ELSE local;
  sH.channelState ← active;
  sH.waitTime ← MilliSecondsToPulses[Socket.defaultWaitTime];
  -- initialize the condition variables and queues
  BufferDefs.QueueInitialize[@sH.completedReceiveQueue];
  BufferDefs.QueueInitialize[@sH.completedUserGetQueue];
  BufferDefs.QueueInitialize[@sH.pendingUserGetQueue];
  Process.InitializeCondition[
    CommUtilDefs.MaybeShorten[@sH.newReceiveInput],
    Process.MsecToTicks[Inline.LowHalf[Socket.defaultWaitTime]] + 1];
  Process.InitializeCondition[
    CommUtilDefs.MaybeShorten[@sH.newUserInput],
    Process.MsecToTicks[Inline.LowHalf[Socket.defaultWaitTime]] + 1];
  -- tell the router of this socket
  Router.AddSocket[sH];
  END; -- CreateInternal

-- This procedure returns the buffer pool for this socket.  It needn't lock the monitor.

GetBufferPool: PUBLIC PROCEDURE [sH: SocketHandle]
  RETURNS [BufferDefs.BufferPool] = BEGIN RETURN[@sH.poolObject]; END;
-- GetBufferPool

-- Cold Procedures

-- This procedure turns this module on.  The Communication software is written with the
-- idea of eventually turning the code on and off during execution, and so we should
-- get the latest value the hostID when being turned on by the Router.

SocketOn: PUBLIC PROCEDURE =
  BEGIN
  -- find  myHostID from the Router
  myHostID ← Router.FindMyHostID[];
  pulsesPerMilliSecond ← LOOPHOLE[System.MicrosecondsToPulses[1000], LONG
    CARDINAL];
  END; -- SocketOn

-- initialization (Cold)

END. -- SocketImpl module
```

LOG

Time: January 7, 1980 3:11 PM By: Dalal Action: converted to new SocketInternal.
Time: January 21, 1980 5:47 PM By: Dalal Action: all sockets under one lock.
Time: March 12, 1980 5:32 PM By: BLyon Action: added send, receive to CreateInternal.

Time: March 18, 1980 12:07 PM By: BLyon Action: Added Reset, Getpacket, PutPacket (body ide
***nticle to old Put), modified Put. Modified TransferWait & TransferWaitAny to abort like PutPacket*
*. **. Modified Abort.*
Time: June 3, 1980 3:43 PM By: BLyon Action: Replace SimpleHeap references with Heap.
Time: June 3, 1980 3:43 PM By: HGM Action: Add MaybeShortens.
Time: June 18, 1980 2:44 PM By: BLyon Action: Added EXPORTED TYPES.
Time: September 18, 1980 3:45 PM By: BLyon Action: flushed primaryNetID concept.
Time: October 8, 1980 9:10 AM By: BLyon Action: Time units are now pulses and not millisecond
***s.*

```
-- File: StatsHot.mesa, Last Edit: HGM June 12, 1980 12:53 PM

DIRECTORY
  StatsDefs USING [StatCounterIndex],
  StatsOps USING [];

StatsHot: PROGRAM EXPORTS StatsDefs, StatsOps =
  BEGIN OPEN StatsDefs;

  -- It would be nice if this were a MONITOR (locking StatsOps.statLock), but that slows things do
**wn a lot and complicates binding in the Pilot world.

  -- These are the master counters. They are the only ones that get bumped by StatIncr and StatB
**ump. The numbers that StatSince prints out are obtained by subtracting two copies of statGran
**d.

  statGrand: PUBLIC ARRAY StatCounterIndex OF LONG CARDINAL;

  StatIncr: PUBLIC PROCEDURE [which: StatCounterIndex] =
    BEGIN statGrand[which] ← statGrand[which] + 1; END;

  StatBump: PUBLIC PROCEDURE [which: StatCounterIndex, howmuch: CARDINAL] =
    BEGIN statGrand[which] ← statGrand[which] + howmuch; END;

  END.
```

```
-- SystemBufferPoolImpl.mesa (last edited by: Garlick on: October 12, 1980 4:53 PM)

DIRECTORY
  BufferDefs USING [
    defaultSystemBufferPoolSize, defaultSystemBuffersToReserve,
    defaultDataWordsPerSystemBuffer, BufferPool, BufferPoolObject, Buffer,
    PupBuffer, RppBuffer, OisBuffer, SppBuffer, BufferObject, Dequeue, Enqueue,
    Queue, QueueInitialize, QueueCleanup],
  CommUtilDefs USING [
    AllocateBuffers, FreeBuffers, LockBuffers, UnlockBuffers, Allocatelocbs,
    Freelocbs, GetReturnFrame, MaybeShorten],
  DriverTypes USING [bufferSeal, bufferPoolSeal, queueSeal],
  DriverDefs USING [
    BufferPoolImpl, QueueImpl, doDebug, doSee, doStats, Glitch, GetGiantVector],
  Inline USING [BITAND, LowHalf],
  OISCPDefs, -- EXPORTS
  Process USING [InitializeCondition, MsecToTicks],
  PupDefs USING [DequeuePup, EnqueuePup, DequeueRpp, EnqueueRpp],
  StatsDefs USING [StatIncr];

SystemBufferPoolImpl: MONITOR
  IMPORTS
    Inline, Process, BufferDefs, CommUtilDefs, DriverDefs, PupDefs, StatsDefs
  EXPORTS BufferDefs, DriverDefs, OISCPDefs, PupDefs
  SHARES BufferDefs =
  BEGIN OPEN BufferDefs, DriverDefs;

  -- monitor protected data
  systemBufferPool: BufferPool ← @systemBufferPoolObject;
  -- systemBufferPoolObject: BufferPoolObject; + + delcared below when initialized
  systemBufferSize: CARDINAL;
  useCount: CARDINAL ← 0;
  wordsPerlocb: CARDINAL ← 0;
  systemBufferPoolSize: CARDINAL ← BufferDefs.defaultSystemBufferPoolSize;

  -- Buffers and IOCB locations must be rounded up for alignment constraints.
  -- Alto SLA Microcode needs IOCB/LCB to be EVEN.
  -- D0 Ethernet/Xwire Microcode needs IOCB to be Quad word aligned,
  -- and first data word to be (almost?) QUAD word aligned.

  -- size of a buffer without any data
  rawOverhead: CARDINAL = SIZE[raw BufferDefs.BufferObject];
  -- 1 extra for pup checksum
  pupOverhead: CARDINAL = SIZE[pupWords pup BufferDefs.BufferObject] + 1;
  overhead: CARDINAL = MAX[
    pupOverhead, SIZE[rppWords rpp BufferDefs.BufferObject] + 1, SIZE[oisWords ois
      BufferDefs.BufferObject], SIZE[sppWords spp ois BufferDefs.BufferObject]];

  -- for the Glitches
  QueueSealBroken: PUBLIC ERROR = CODE;
  PoolSealBroken: PUBLIC ERROR = CODE;
  BufferSealBroken: PUBLIC ERROR = CODE;
  FreeQueueNotInitialized: PUBLIC ERROR = CODE;
  QueueScrambled: PUBLIC ERROR = CODE;
  CantResetWhileActive: PUBLIC ERROR = CODE;
  BufferPoolSealBroken: PUBLIC ERROR = CODE;

  -- Cold Procedures
```



```
  AdjustBufferParms: PUBLIC ENTRY PROCEDURE [
    bufferPoolSize, bufferSize: CARDINAL] =
    BEGIN
    IF useCount # 0 THEN Glitch[CantResetWhileActive];
    IF bufferPoolSize # 0 THEN systemBufferPoolSize ← bufferPoolSize;
    IF bufferSize # 0 THEN systemBufferPool.dataWordsPerBuffer ← bufferSize;
    END;

  GetBufferParms: PUBLIC PROCEDURE
    RETURNS [bufferPoolSize, bufferSize: CARDINAL] =
    BEGIN
    IF useCount # 0 THEN
      RETURN[systemBufferPool.total, systemBufferPool.dataWordsPerBuffer];
    RETURN[systemBufferPoolSize, systemBufferPool.dataWordsPerBuffer];
    END;

  GetWordsPerlocb: PUBLIC PROCEDURE RETURNS [CARDINAL] =
    BEGIN RETURN[wordsPerlocb]; END;

  SetWordsPerlocb: PUBLIC PROCEDURE [new: CARDINAL] =
    BEGIN wordsPerlocb ← new; END;

  -- NB: wordsPerBuffer and wordsPerlocb MUST be QuadWord multiples

  FreeQueueMake: PUBLIC PROCEDURE [extra: CARDINAL] =
    BEGIN OPEN systemBufferPool;
    b: Buffer;
    x: --SHORT--POINTER TO BufferObject = NIL;
    i: CARDINAL;
    useCount ← useCount + 1;
    -- 2 for checksum, 4 for end test, 3 for round down
    wordsPerBuffer ← dataWordsPerBuffer + overhead + 2 + 4 + 3;
    UNTIL Inline.BITAND[wordsPerBuffer, 3] = 0 DO
      wordsPerBuffer ← wordsPerBuffer + 1; ENDLOOP;
    -- systemBufferSize is a little bigger because of the quad word allignment
    systemBufferSize ← wordsPerBuffer - (@x.encapsulation - LOOPHOLE[x, POINTER]);
    total ← systemBufferPoolSize + extra;
    firstBuffer ← CommUtilDefs.AllocateBuffers[wordsPerBuffer*total + 3];
    CommUtilDefs.LockBuffers[firstBuffer];
    -- This determines the buffer alignment: see DriverTypes.Encapsulation
    UNTIL Inline.BITAND[Inline.LowHalf[@firstBuffer.encapsulation], 3] = 3 DO
      firstBuffer ← firstBuffer + 1; ENDLOOP;
    IF doDebug THEN
      BEGIN
      GetGiantVector[].firstBuffer ← firstBuffer;
      GetGiantVector[].wordsPerBuffer ← wordsPerBuffer;
      GetGiantVector[].bufferPoolSize ← total;
      END;
    QueueInitialize[@freeQueue];
    b ← firstBuffer;
    FOR i IN [0..total) DO
      Process.InitializeCondition[
        CommUtilDefs.MaybeShorten[@b.sendCompleted], Process.MsecToTicks[10000]];
      b.iocbChain ← NIL;
      b.userData ← NIL;
      b.userPtr ← NIL;
      b.allNets ← b.bypassZeroNet ← FALSE;
      b.type ← raw;
```

```
    b.size ← big;
    b.pupLength ← b.length ← 0;
    b.bufferNumber ← i;
    b.pupType ← last;
    b.queue ← NIL;
    b.pool ← systemBufferPool;
    b.next ← NIL;
    IF doDebug THEN b.seal ← DriverTypes.bufferSeal;
    b.requeueProcedure ← ReturnFreeBuffer;
    Enqueue[@freeQueue, b];
    b ← b + wordsPerBuffer;
    ENDLOOP;
  IF wordsPerIocb # 0 THEN
    BEGIN
    iocb: LONG POINTER;
    iocb ← CommUtilDefs.AllocateIocbs[wordsPerIocb*total + 3];
    b ← firstBuffer;
    FOR i IN [0..total) DO
      -- this does NOT align each individual iocb
      b.iocbChain ← iocb;
      iocb ← iocb + wordsPerIocb;
      b ← b + wordsPerBuffer;
      ENDLOOP;
    END;
  END;

FreeQueueDestroy: PUBLIC PROCEDURE =
  BEGIN
  UNTIL Dequeue[@systemBufferPool.freeQueue] = NIL DO ENDLOOP;
  QueueCleanup[@systemBufferPool.freeQueue];
  CommUtilDefs.UnlockBuffers[systemBufferPool.firstBuffer];
  IF systemBufferPool.firstBuffer.iocbChain # NIL THEN
    CommUtilDefs.FreeIocbs[systemBufferPool.firstBuffer.iocbChain];
  CommUtilDefs.FreeBuffers[systemBufferPool.firstBuffer];
  systemBufferPool.firstBuffer ← NIL;
  IF doDebug THEN GetGiantVector[].firstBuffer ← NIL;
  useCount ← 0;
  END;

-- Cool Procedures

GetSystemBufferPool: PUBLIC PROCEDURE RETURNS [BufferDefs.BufferPool] =
  BEGIN RETURN[systemBufferPool]; END;

-- This is not an ENTRY procedure because we assume that systemBufferPool can
-- have its parameters set only once.

DataWordsPerPupBuffer: PUBLIC PROCEDURE RETURNS [CARDINAL] =
  BEGIN
  RETURN[systemBufferPool.dataWordsPerBuffer + (overhead - pupOverhead)];
  END;

-- This is not an ENTRY procedure because we assume that systemBufferPool can
-- have its parameters set only once.

DataWordsPerRawBuffer: PUBLIC PROCEDURE RETURNS [CARDINAL] =
  BEGIN
  RETURN[systemBufferPool.dataWordsPerBuffer + (overhead - rawOverhead)];
```



```
  END;

-- This is not an ENTRY procedure because we assume that systemBufferPool can
-- have its parameters set only once.

DataWordsPerOisBuffer: PUBLIC PROCEDURE RETURNS [CARDINAL] =
  BEGIN
  RETURN[
    systemBufferPool.dataWordsPerBuffer +
      (overhead - SIZE[oisWords ois BufferDefs.BufferObject])];
  END;

-- This is not an ENTRY procedure because we assume that systemBufferPool can
-- have its parameters set only once.

DataWordsPerSppBuffer: PUBLIC PROCEDURE RETURNS [CARDINAL] =
  BEGIN
  RETURN[
    systemBufferPool.dataWordsPerBuffer +
      (overhead - SIZE[sppWords spp ois BufferDefs.BufferObject])];
  END;

-- Hot Procedures

GetFreeBuffer: PUBLIC ENTRY PROCEDURE RETURNS [b: Buffer] =
  BEGIN
  ENABLE UNWIND => NULL;
  IF doDebug AND systemBufferPool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  IF doStats AND ~systemBufferPool.freeQueue.length > systemBufferPool.reserve
    THEN StatsDefs.StatIncr[statBufferWaits];
  UNTIL systemBufferPool.freeQueue.length > systemBufferPool.reserve DO
    WAIT systemBufferPool.freeQueueNotEmpty; ENDLOOP;
  b ← Dequeue[@systemBufferPool.freeQueue];
  IF doDebug AND b = NIL THEN Glitch[QueueScrambled];
  IF doDebug AND b.pool # systemBufferPool THEN Glitch[QueueScrambled];
  IF doDebug OR doSee THEN b.type ← raw;
  IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
  END;

-- Get free buffer, but don't wait if there is none
-- NB: These will return the last buffer, use with caution

MaybeGetFreeBuffer: PUBLIC ENTRY PROCEDURE RETURNS [b: Buffer] =
  BEGIN
  IF doDebug AND systemBufferPool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  IF doStats AND systemBufferPool.freeQueue.length = 0 THEN
    StatsDefs.StatIncr[statBufferWaits];
  b ← Dequeue[@systemBufferPool.freeQueue];
  IF b # NIL THEN
    BEGIN
    IF doDebug AND b.pool # systemBufferPool THEN Glitch[QueueScrambled];
    IF doDebug OR doSee THEN b.type ← raw;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
```

```
    END;
  END;

ReturnFreePupBuffer: PUBLIC PROCEDURE [PupBuffer] = LOOPHOLE[ReturnFreeBuffer];
ReturnFreeRppBuffer: PUBLIC PROCEDURE [RppBuffer] = LOOPHOLE[ReturnFreeBuffer];
ReturnFreeOisBuffer: PUBLIC PROCEDURE [OisBuffer] = LOOPHOLE[ReturnFreeBuffer];
ReturnFreeSppBuffer: PUBLIC PROCEDURE [SppBuffer] = LOOPHOLE[ReturnFreeBuffer];
ReturnFreeBuffer: PUBLIC ENTRY PROCEDURE [b: Buffer] =
  BEGIN
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  IF doDebug AND b.pool # systemBufferPool THEN Glitch[QueueScrambled];
  -- Note: we do some "initialization" of things here since there are several ways to get
  -- buffers from the freeQueue.
  b.requeueProcedure ← ReturnFreeBuffer;
  b.network ← NIL;
  b.userPtr ← b.userData ← NIL;
  Enqueue[@systemBufferPool.freeQueue, b];
  -- This is ugly, but there is'nt any way to make things work without it, if 2
  -- PROCESSes are waiting for buffers.
  BROADCAST systemBufferPool.freeQueueNotEmpty;
  END;

BuffersLeft: PUBLIC ENTRY PROCEDURE RETURNS [left: CARDINAL] =
  BEGIN
  IF doDebug AND systemBufferPool.seal # DriverTypes.bufferPoolSeal THEN
    Glitch[PoolSealBroken];
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  left ← systemBufferPool.freeQueue.length;
  left ←
    (IF left > systemBufferPool.reserve THEN left - systemBufferPool.reserve
     ELSE 0);
  END;

-- This routine is only used by device drivers to get buffers to read things into.
-- NB: b.length is setup for the size of the buffer, including ALL of the encapsulation.
-- All device drivers except for the XWire must fudge things themsleves, since they do
-- not use all of the encapsulation field.

GetInputBuffer: PUBLIC ENTRY PROCEDURE RETURNS [b: Buffer] =
  BEGIN
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  b ← Dequeue[@systemBufferPool.freeQueue];
  IF b # NIL THEN
    BEGIN
    IF doDebug AND b.pool # systemBufferPool THEN Glitch[QueueScrambled];
    b.length ← systemBufferSize;
    IF doDebug OR doSee THEN b.type ← raw;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
    END;
  END;

-- These procedures get buffers from the system buffer pool

GetFreeOisBuffer: PUBLIC PROCEDURE RETURNS [b: OisBuffer] =
  BEGIN
  b ← LOOPHOLE[GetFreeBuffer[], OisBuffer];
```



```
  IF doDebug OR doSee THEN b.type ← ois;
  IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
  END;

MaybeGetFreeOisBuffer: PUBLIC PROCEDURE RETURNS [b: OisBuffer] =
  BEGIN
  b ← LOOPHOLE[MaybeGetFreeBuffer[], OisBuffer];
  IF b # NIL THEN BEGIN IF doDebug OR doSee THEN b.type ← ois; END;
  END;


GetFreePupBuffer: PUBLIC PROCEDURE RETURNS [b: PupBuffer] =
  BEGIN
  b ← LOOPHOLE[GetFreeBuffer[], PupBuffer];
  IF doDebug OR doSee THEN b.type ← pup;
  b.pupType ← data;
  IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
  END;

-- Get free buffer, but don't wait if there is none
-- NB: These will return the last buffer, use with caution

MaybeGetFreePupBuffer: PUBLIC PROCEDURE RETURNS [b: PupBuffer] =
  BEGIN
  b ← LOOPHOLE[MaybeGetFreeBuffer[], PupBuffer];
  IF b # NIL THEN
    BEGIN IF doDebug OR doSee THEN b.type ← pup; b.pupType ← data; END;
  END;

GetClumpOfPupBuffers: PUBLIC ENTRY PROCEDURE [
  q: Queue, n: CARDINAL, wait: BOOLEAN] =
  BEGIN
  ENABLE UNWIND => NULL;
  b: PupBuffer;
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  IF doDebug AND q.seal # DriverTypes.queueSeal THEN Glitch[QueueSealBroken];
  UNTIL systemBufferPool.freeQueue.length > n + systemBufferPool.reserve DO
    IF ~wait THEN RETURN; WAIT systemBufferPool.freeQueueNotEmpty; ENDLOOP;
  THROUGH [0..n) DO
    b ← PupDefs.DequeuePup[@systemBufferPool.freeQueue];
    IF doDebug OR doSee THEN b.type ← pup;
    b.pupType ← data;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
    PupDefs.EnqueuePup[q, b];
    ENDLOOP;
  END;

GetFreeRppBuffer: PUBLIC PROCEDURE RETURNS [b: RppBuffer] =
  BEGIN
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  b ← LOOPHOLE[GetFreeBuffer[], RppBuffer];
  IF doDebug OR doSee THEN b.type ← rpp;
  b.rppPupType ← 4;
  b.system ← b.sendAck ← b.notify ← FALSE;
  IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
  END;
```

```
MaybeGetFreeRppBuffer: PUBLIC PROCEDURE RETURNS [b: RppBuffer] =
  BEGIN
  b ← LOOPHOLE[MaybeGetFreeBuffer[], RppBuffer];
  IF b # NIL THEN
    BEGIN
    IF doDebug OR doSee THEN b.type ← rpp;
    b.rppPupType ← 4;
    b.system ← b.sendAck ← b.notify ← FALSE;
    END;
  END;

GetClumpOfRppBuffers: PUBLIC ENTRY PROCEDURE [
  q: Queue, n: CARDINAL, wait: BOOLEAN] =
  BEGIN
  ENABLE UNWIND => NULL;
  b: RppBuffer;
  IF doDebug AND systemBufferPool.firstBuffer = NIL THEN
    Glitch[FreeQueueNotInitialized];
  IF doDebug AND q.seal # DriverTypes.queueSeal THEN Glitch[QueueSealBroken];
  UNTIL systemBufferPool.freeQueue.length > n + systemBufferPool.reserve DO
    IF ~wait THEN RETURN; WAIT systemBufferPool.freeQueueNotEmpty; ENDLOOP;
  THROUGH [0..n) DO
    b ← PupDefs.DequeueRpp[@systemBufferPool.freeQueue];
    IF doDebug OR doSee THEN b.type ← pup;
    b.rppPupType ← 4;
    b.system ← b.sendAck ← b.notify ← FALSE;
    IF doDebug THEN b.debug ← CommUtilDefs.GetReturnFrame[].accesslink;
    PupDefs.EnqueueRpp[q, b];
    ENDLOOP;
  END;

-- initialization

systemBufferPoolObject: BufferPoolObject ←
  [LOCK:, freeQueue:, seal: DriverTypes.bufferPoolSeal, freeQueueNotEmpty:,
    sendBufferAvailable:, receiveBufferAvailable:, total:,
    reserve: BufferDefs.defaultSystemBuffersToReserve, send: 0, receive: 0,
    sendInUse: 0, receiveInUse: 0,
    dataWordsPerBuffer: BufferDefs.defaultDataWordsPerSystemBuffer,
    firstBuffer: NIL, bufferSize: big, wordsPerBuffer:];
Process.InitializeCondition[
  CommUtilDefs.MaybeShorten[@systemBufferPool.freeQueueNotEmpty],
  Process.MsecToTicks[10000]];
Process.InitializeCondition[
  CommUtilDefs.MaybeShorten[@systemBufferPool.sendBufferAvailable],
  Process.MsecToTicks[10000]];
Process.InitializeCondition[
  CommUtilDefs.MaybeShorten[@systemBufferPool.receiveBufferAvailable],
  Process.MsecToTicks[10000]];

START DriverDefs.QueueImpl;
START DriverDefs.BufferPoolImpl;
IF doDebug THEN
  BEGIN
  GetGiantVector[].firstBuffer ← NIL;
  GetGiantVector[].freeQueue ← @systemBufferPool.freeQueue;
  END;
END. -- SystemBufferPoolImpl module
```


LOG

Time: April 18, 1980 9:16 AM By: Dalal Action: created file for Pilot 5.0.
Time: April 21, 1980 10:29 AM By: Dalal Action: merged all buffer routines.
Time: May 8, 1980 1:19 PM By: BLyon Action: Moved wordsPerlocb declaration/initialization to B
***oss.*
Time: September 14, 1980 2:35 AM By: HGM Action: Add parm to FreeQueueMake, and correspo
***nding fiddles to keep track of size of systemBufferPool.*
Time: October 12, 1980 4:50 PM By: Garlick Action: Made change in FreeQueueMake to set total
***buffers to be based on BufferDefs.defaultSystemBufferPoolSize (with no compensation)*

```
-- UserTerminalImpl.mesa (last edited by: McJones on: August 28, 1980 11:44 AM)
-- This is currently crocked to use CreateForCode instead of Create to avoid a bug in the CSL displ
**ay microcode that can't cross 64K boundaries!!

DIRECTORY
  BitBlt USING [BBTable],
  DisplayFace USING [
    Connect, Cursor, CursorPtr, cursorPosition, Disconnect, GetBitBltTable,
    globalStateSize, hasBorder, hasBuffer, height, Initialize, InitializeCleanup,
    pagesForBitmap, pixelsPerInch, SetBackground, SetBorderPattern,
    SetCursorPattern, TurnOn, TurnOff, width],
  DriverStartChain USING [Start],
  Environment USING [PageCount, PageNumber, PageOffset],
  KeyboardFace USING [keyboard],
  MouseFace USING [position, SetPosition],
  PilotSwitches USING [switches --.u--],
  Process USING [MsecToTicks, SetTimeout],
  ProcessInternal USING [AllocateNakedCondition],
  ResidentHeap USING [MakeNode],
  Runtime USING [CallDebugger],
  Space USING [
    Create, defaultBase, defaultWindow, Delete, Handle, LongPointer, Map,
    nullHandle, virtualMemory, VMPageNumber],
  SpecialSpace USING [CreateForCode, MakeResident, MakeSwappable],
  System USING [CreateIntervalTimer, GetIntervalTime, Microseconds, TimerHandle],
  UserTerminal,
  Transaction USING [],
  Zone USING [Base, Status];

UserTerminalImpl: MONITOR
  IMPORTS
    DisplayFace, OtherDrivers: DriverStartChain, KeyboardFace, MouseFace,
    PilotSwitches, Process, ProcessInternal, ResidentHeap, Runtime, Space,
    SpecialSpace, System, Transaction
  EXPORTS DriverStartChain, UserTerminal
  SHARES UserTerminal =

  BEGIN OPEN UserTerminal;

  -- Constants (modulo display hardware and processor)

  pixelsPerInch: PUBLIC CARDINAL ← DisplayFace.pixelsPerInch;
  screenHeight: PUBLIC CARDINAL [0..32767] ← DisplayFace.height;
  screenWidth: PUBLIC CARDINAL [0..32767] ← DisplayFace.width;

  mouse: PUBLIC LONG POINTER TO READONLY Coordinate ←
    LOOPHOLE[MouseFace.position];
  cursor: PUBLIC LONG POINTER TO Coordinate ←
    LOOPHOLE[DisplayFace.cursorPosition];
  keyboard: PUBLIC LONG POINTER TO READONLY ARRAY OF WORD ←
    LOOPHOLE[KeyboardFace.keyboard];
  hasBorder: PUBLIC BOOLEAN ← DisplayFace.hasBorder;

  state: State ← disconnected;

  GetState: PUBLIC ENTRY PROC RETURNS [State] = {RETURN[state]};

  background: Background ← white;
```



```
  GetBackground: PUBLIC ENTRY PROC RETURNS [Background] = {RETURN[background]};

  bitmapBase: LONG POINTER ← NIL;

  BitmapIsDisconnected: PUBLIC ERROR = CODE;

  GetBitBltTable: PUBLIC ENTRY PROC RETURNS [BitBlt.BBTable] =
    BEGIN
    IF state = disconnected THEN RETURN WITH ERROR BitmapIsDisconnected;
    RETURN[DisplayFace.GetBitBltTable[]]
    END;

  SetMousePosition: PUBLIC ENTRY PROC [newMousePosition: Coordinate] = {
    MouseFace.SetPosition[[newMousePosition.x, newMousePosition.y]]};

  saveCursor: ARRAY [0..16) OF WORD ← ALL[0];

  GetCursorPattern: PUBLIC ENTRY PROC RETURNS [cursorPattern: CursorArray] = {
    RETURN[saveCursor]};

  SetCursorPattern: PUBLIC ENTRY PROC [cursorPattern: CursorArray] =
    BEGIN
    saveCursor ← cursorPattern;
    DisplayFace.SetCursorPattern[@saveCursor];
    END;

  SetBorder: PUBLIC PROCEDURE [oddPairs, evenPairs: [0..377B]] = {
    DisplayFace.SetBorderPattern[oddPairs, evenPairs]};

  SetBackground: PUBLIC ENTRY PROC [new: Background] RETURNS [old: Background] =
    BEGIN
    IF (old ← background) = new THEN RETURN;
    DisplayFace.SetBackground[
      IF (background ← new) = white THEN white ELSE black];
    END;

  bitmapSpace: Space.Handle ← Space.nullHandle;

  SetState: PUBLIC ENTRY PROC [new: State] RETURNS [old: State] =
    BEGIN
    IF (old ← state) = new THEN RETURN;
    IF old = disconnected THEN
      BEGIN
      swapUnitSize: Environment.PageCount = 50;
      bitmapSpace ← SpecialSpace.CreateForCode[
        size: DisplayFace.pagesForBitmap, parent: Space.virtualMemory,
        base: Space.defaultBase];
      IF PilotSwitches.switches.u = up THEN
        FOR offset: CARDINAL ← 0, offset + swapUnitSize WHILE
          offset + swapUnitSize <= DisplayFace.pagesForBitmap DO
          [] ← Space.Create[size: swapUnitSize, parent: bitmapSpace, base: offset]
          ENDLOOP;
      IF ~DisplayFace.hasBuffer THEN
        Space.Map[space: bitmapSpace, window: Space.defaultWindow];
      bitmapBase ← Space.LongPointer[bitmapSpace];
      DisplayFace.Connect[Space.VMPageNumber[bitmapSpace]];
      IF new = (state ← off) THEN RETURN;
      END; -- assert state = on or off, and state # new
    SELECT new FROM
```

```
    on =>
      BEGIN
      IF ~DisplayFace.hasBuffer THEN SpecialSpace.MakeResident[bitmapSpace];
      DisplayFace.TurnOn[];
      END;
    off, disconnected =>
      BEGIN
      DisplayFace.TurnOff[];
      WaitForVerticalRetrace[];
      WaitForVerticalRetrace[];
      IF ~DisplayFace.hasBuffer THEN SpecialSpace.MakeSwappable[bitmapSpace];
      END;
    ENDCASE => ERROR;
  IF new = disconnected THEN
    BEGIN
    DisplayFace.Disconnect[];
    Space.Delete[bitmapSpace];
    bitmapSpace ← Space.nullHandle;
    bitmapBase ← NIL;
    END;
  state ← new;
  END;

hardwareVerticalRetrace: LONG POINTER TO CONDITION;
verticalRetrace, blinkTime: CONDITION;
meFirst: BOOLEAN ← TRUE;
microsecondsPerScanLine: CARDINAL = 14; -- should be in terms of DisplayFace

BlinkDisplay: PUBLIC ENTRY PROCEDURE =
  BEGIN OPEN DisplayFace;
  SetBackground[IF background = white THEN black ELSE white]; -- invert display
  WAIT blinkTime;
  SetBackground[IF background = white THEN white ELSE black]; -- put it back

  END;

-- Miscellaneous

WaitForScanLine: PUBLIC PROCEDURE [scanLine: INTEGER] =
  BEGIN
  WaitForScanLineZero: ENTRY PROCEDURE = INLINE
    BEGIN WaitForVerticalRetrace[] END;
  WaitForScanLineZero[];
  IF scanLine = 0 OR CARDINAL[scanLine] > screenHeight THEN NULL
  ELSE
    BEGIN OPEN System;
    timer: TimerHandle = CreateIntervalTimer[];
    interval: Microseconds = scanLine*microsecondsPerScanLine;
    UNTIL GetIntervalTime[timer] >= interval DO ENDLOOP
    END
  END;

WaitForVerticalRetrace: INTERNAL PROCEDURE =
  BEGIN
  IF meFirst THEN
    BEGIN
    meFirst ← FALSE;
    WAIT hardwareVerticalRetrace;
    BROADCAST verticalRetrace;
```



```
    meFirst ← TRUE
    END
  ELSE WAIT verticalRetrace
  END;

-- Initialization

Start: PUBLIC PROCEDURE =
  BEGIN
  mask: WORD;
  headGlobalP: Zone.Base RELATIVE POINTER;
  s: Zone.Status;
  [headGlobalP, s] ← ResidentHeap.MakeNode[DisplayFace.globalStateSize, a16];
  IF s # okay THEN Runtime.CallDebugger["Zone error in UserTerminalImpl"L];
  [cv: hardwareVerticalRetrace, mask: mask] ←
    ProcessInternal.AllocateNakedCondition[];
  DisplayFace.Initialize[headGlobalP, mask];
  DisplayFace.InitializeCleanup[];
  Process.SetTimeout[@blinkTime, Process.MsecToTicks[100]];
  OtherDrivers.Start[];
  END;

END.
```

July 31, 1980 9:54 AM *McJones* *Use pagesForBitmap, GetBitBltTable from DisplayFace*
August 28, 1980 11:44 AM *McJones* *Convert to ProcessInternal.AllocateNakedCondition*

```
-- DiskChannelImpl.mesa (last edited by: Luniewski on: October 3, 1980 12:07 PM)

DIRECTORY
  Device USING [nullType, Type],
  DiskChannel USING [
    Address, CompletionHandle, DiskPageCount, DiskPageNumber, Drive, DriveState,
    DriveStatus, Handle, IORequestHandle, nullDrive, nullHandle],
  DiskChannelBackend USING [DriveHandle, DriveID],
  Environment USING [Base, first64K],
  PhysicalVolume USING [nullDeviceIndex],
  PilotDisk USING [Handle],
  Process USING [InitializeCondition, MsecToTicks, SetTimeout],
  ResidentHeap USING [FreeNode, MakeNode],
  RuntimeInternal USING [WorryCallDebugger],
  Zone USING [BlockSize, Status];

DiskChannelImpl: MONITOR
  IMPORTS Process, ResidentHeap, RuntimeInternal
  EXPORTS DiskChannel, DiskChannelBackend
  SHARES DiskChannel, DiskChannelBackend, PilotDisk =
  BEGIN

  -- Miscellaneous
  base: Environment.Base = Environment.first64K;
  ErrorHalt: PROC = {
    RuntimeInternal.WorryCallDebugger["Error in DiskChannelImpl"L]; };

  driveObjectUnlocked: CONDITION;
  g: GlobalHandle ← MakeLP[SIZE[Globals]];
  GlobalHandle: TYPE = LONG POINTER TO Globals;
  Globals: TYPE = RECORD [
    firstDrive: DiskChannelBackend.DriveHandle,
    -- head of the linked list of known drives
    channelRestarted: CONDITION]; -- used in the implementation of Restart

  MakeLP: PROC [size: Zone.BlockSize] RETURNS [lp: LONG POINTER] =
    BEGIN
    t: Environment.Base RELATIVE POINTER = MakeRP[size];
    RETURN[@base[t]];
    END;

  MakeRP: PROC [size: Zone.BlockSize]
    RETURNS [rp: Environment.Base RELATIVE POINTER] =
    BEGIN
    status: Zone.Status;
    [node: rp, s: status] ← ResidentHeap.MakeNode[size, a1];
    IF status ~= okay THEN ErrorHalt[];
    END;

  -- Drives

  FindDrive: PROC [drive: DiskChannel.Drive]
    RETURNS [driveH: DiskChannelBackend.DriveHandle] = INLINE {
    RETURN[LOOPHOLE[drive]]};

  shortTime: CONDITION;

  AwaitStateChange: PUBLIC ENTRY PROC [
    count: CARDINAL, type: Device.Type, index: CARDINAL]
```



```
    RETURNS [currentChangeCount: CARDINAL] =
    BEGIN
    DriveOK: PROC [drive: DiskChannelBackend.DriveHandle] RETURNS [BOOLEAN] =
      BEGIN
      IF type = Device.nullType THEN
        RETURN[
          index = drive.driveOrdinal OR index = PhysicalVolume.nullDeviceIndex]
      ELSE
        RETURN[
          type = drive.driveID.type AND
            (index = drive.driveOrdinal OR index =
              PhysicalVolume.nullDeviceIndex)]
      END;
    GetCount: PROC RETURNS [count: CARDINAL] =
      BEGIN
      drive: DiskChannelBackend.DriveHandle ← g.firstDrive;
      count ← 0;
      WHILE drive ~= NIL DO
        IF DriveOK[drive] THEN
          BEGIN -- we must poke drive to be sure that drive.changeCount is right
          [] ← drive.getStatus[drive];
          count ← count + drive.changeCount;
          END;
        drive ← drive.next;
        ENDLOOP
      END;
    WHILE (currentChangeCount ← GetCount[]) < count DO WAIT shortTime ENDLOOP;
    END;

  GetDriveAttributes: PUBLIC ENTRY PROC [drive: DiskChannel.Drive]
    RETURNS [
      deviceType: Device.Type, deviceHandle: PilotDisk.Handle,
      deviceOrdinal: CARDINAL, nPages: DiskChannel.DiskPageCount, ready: BOOLEAN,
      state: DiskChannel.DriveState, changeCount: CARDINAL] =
    BEGIN
    driveID: DiskChannelBackend.DriveID;
    ready ← FindDrive[drive].getStatus[FindDrive[drive]].ready;
    [driveID: driveID, nPages: nPages, driveOrdinal: deviceOrdinal, state: state,
      changeCount: changeCount] ← FindDrive[drive]↑;
    deviceType ← driveID.type;
    deviceHandle ← driveID.handle;
    END;

  GetDriveTag: PUBLIC ENTRY PROC [drive: DiskChannel.Drive] RETURNS [CARDINAL] = {
    RETURN[FindDrive[drive].tag]};

  GetNextDrive: PUBLIC ENTRY PROC [prev: DiskChannel.Drive]
    RETURNS [DiskChannel.Drive] =
    BEGIN
    IF prev = DiskChannel.nullDrive THEN RETURN[SealDrive[g.firstDrive]]
    ELSE RETURN[SealDrive[FindDrive[prev].next]]
    END;

  GetPageNumber: PUBLIC ENTRY PROC [
    drive: DiskChannel.Drive, page: DiskChannel.Address]
    RETURNS [DiskChannel.DiskPageNumber] =
    BEGIN
    dH: DiskChannelBackend.DriveHandle = FindDrive[drive];
    RETURN[dH.getPageNumber[dH, page]];
```

```
  END;

SealDrive: PROC [driveID: DiskChannelBackend.DriveHandle]
  RETURNS [DiskChannel.Drive] = INLINE {RETURN[LOOPHOLE[driveID]]};

SetDriveState: PUBLIC PROC [
  drive: DiskChannel.Drive, changeCount: CARDINAL,
  state: DiskChannel.DriveState] RETURNS [s: DiskChannel.DriveStatus] =
  BEGIN
  AcquireDriveObject: ENTRY PROCEDURE [drive: DiskChannelBackend.DriveHandle] =
    BEGIN
    WHILE drive.stateChangeLocked DO WAIT driveObjectUnlocked; ENDLOOP;
    drive.stateChangeLocked ← TRUE;
    -- poke the drive so that drive.changeCount is current
    [] ← drive.getStatus[drive];
    END;
  ReturnDriveObject: ENTRY PROCEDURE [drive: DiskChannelBackend.DriveHandle] = {
    drive.stateChangeLocked ← FALSE; BROADCAST driveObjectUnlocked; };
  handle: DiskChannelBackend.DriveHandle = FindDrive[drive];
  AcquireDriveObject[handle];
  -- Special case changeCount of 0 for PhysicalVolumeImpl
  IF changeCount ~= 0 THEN
    IF changeCount ~= handle.changeCount THEN RETURN[invalidDrive];
  IF NOT (handle.state = inactive OR state = inactive) THEN {
    ReturnDriveObject[handle]; RETURN[alreadyAsserted]; };
  handle.changeState[handle, state];
  handle.state ← state;
  ReturnDriveObject[handle];
  RETURN[ok]
  END;

SetDriveTag: PUBLIC ENTRY PROC [drive: DiskChannel.Drive, tag: CARDINAL] = {
  FindDrive[drive].tag ← tag};

-- Channels

ChannelHandle: TYPE = Environment.Base RELATIVE POINTER TO ChannelObject;
ChannelObject: TYPE = RECORD [
  driveHandle: DiskChannelBackend.DriveHandle,
  completionHandle: CompletionHandle, -- used in the implementation of WaitAny
  changeCount: CARDINAL,
  suspendCount: [0..256), -- used in the implementation of Suspend, Restart
  ioCount: [0..256)]; -- used in the implementation of Idle
Create: PUBLIC ENTRY PROC [
  drive: DiskChannel.Drive, completion: DiskChannel.CompletionHandle]
  RETURNS [DiskChannel.Handle] =
  BEGIN OPEN comp: LOOPHOLE[completion, CompletionHandle];
  chH: ChannelHandle;
  dH: DiskChannelBackend.DriveHandle = FindDrive[drive];
  IF dH = NIL THEN RETURN[LOOPHOLE[DiskChannel.nullHandle]]; -- no such drive
  chH ← MakeRP[SIZE[ChannelObject]];
  base[chH] ← ChannelObject[
    driveHandle: dH, completionHandle: @comp, changeCount: dH.changeCount,
    suspendCount: 0, ioCount: 0];
  RETURN[LOOPHOLE[chH]]
  END;

Delete: PUBLIC ENTRY PROC [channel: DiskChannel.Handle] = {
```



```
  IF ResidentHeap.FreeNode[LOOPHOLE[channel]] # okay THEN ErrorHalt[]};

GetAttributes: PUBLIC ENTRY PROC [channel: DiskChannel.Handle]
  RETURNS [drive: DiskChannel.Drive] = {RETURN[SealDrive[GetDrive[channel]]]};

GetPageAddress: PUBLIC ENTRY PROC [
  channel: DiskChannel.Handle, page: DiskChannel.DiskPageNumber]
  RETURNS [DiskChannel.Address] =
  BEGIN OPEN channelObj: base[LOOPHOLE[channel, ChannelHandle]];
  RETURN[channelObj.driveHandle.getPageAddress[channelObj.driveHandle, page]];
  END;

Idle: PUBLIC ENTRY PROC [channel: DiskChannel.Handle] =
  BEGIN OPEN channelObj: base[LOOPHOLE[channel, ChannelHandle]];
  -- Should an error be returned if the change counts do not match?
  IF channelObj.changeCount ~= channelObj.driveHandle.changeCount THEN RETURN;
  channelObj.suspendCount ← channelObj.suspendCount + 1;
  WHILE channelObj.ioCount > 0 DO
    WAIT channelObj.completionHandle.ioComplete; ENDLOOP;
  END;

InitiateIO: PUBLIC --EXTERNAL--PROC [req: DiskChannel.IORequestHandle] =
  BEGIN
  TouchChannel: ENTRY PROC [channelHandle: ChannelHandle] =
    -- do the monitored parts of setting up an IO request.
    -- INLINE
    BEGIN OPEN channelObj: base[channelHandle];
    WHILE channelObj.suspendCount > 0 DO WAIT g.channelRestarted; ENDLOOP;
    channelObj.ioCount ← channelObj.ioCount + 1;
    END;
  CancelIO: ENTRY PROC [channelHandle: ChannelHandle] = INLINE
    BEGIN OPEN channelObj: base[channelHandle];
    channelObj.ioCount ← channelObj.ioCount - 1;
    END;
  ch: ChannelHandle = LOOPHOLE[req.channel];
  TouchChannel[ch];
  IF base[ch].changeCount ~= base[ch].driveHandle.changeCount THEN
    BEGIN
    CancelIO[ch];
    req.status ← invalidChannel;
    NotifyIOComplete[req];
    RETURN;
    END;
  req.countDone ← 0;
  base[ch].driveHandle.requestIO[req];
  END;

Restart: PUBLIC ENTRY PROC [channel: DiskChannel.Handle] =
  BEGIN OPEN channelObj: base[LOOPHOLE[channel, ChannelHandle]];
  IF (channelObj.suspendCount ← channelObj.suspendCount - 1) = 0 THEN
    BROADCAST g.channelRestarted;
  END;

Suspend: PUBLIC ENTRY PROC [channel: DiskChannel.Handle] =
  BEGIN OPEN channelObj: base[LOOPHOLE[channel, ChannelHandle]];
  channelObj.suspendCount ← channelObj.suspendCount + 1;
  END;
```

```
WaitAny: PUBLIC ENTRY PROC [completion: DiskChannel.CompletionHandle]
  RETURNS [req: DiskChannel.IORequestHandle] =
  BEGIN OPEN LOOPHOLE[completion, CompletionHandle];
  WHILE first = NIL DO WAIT ioComplete; ENDLOOP;
  req ← first;
  first ← req.next;
  END;

-- Completions

CompletionHandle: TYPE = LONG POINTER TO CompletionObject;
CompletionObject: TYPE = RECORD [
  first, last: DiskChannel.IORequestHandle, ioComplete: CONDITION];
ioCompletePrototype: CONDITION; -- do not ever wait on this
CreateCompletionObject: PUBLIC PROC RETURNS [DiskChannel.CompletionHandle] =
  BEGIN
  Crock: TYPE = ARRAY [0..SIZE[CONDITION]) OF WORD;
  cH: CompletionHandle ← MakeLP[SIZE[CompletionObject]];
  cH.first ← NIL; -- cH.ioComplete ← ioCompletePrototype;
  LOOPHOLE[@cH.ioComplete, LONG POINTER TO Crock]↑ ←
    LOOPHOLE[ioCompletePrototype];
  RETURN[LOOPHOLE[cH]]
  END;

-- Backend

GetDrive: PUBLIC PROC [channel: DiskChannel.Handle]
  RETURNS [DiskChannelBackend.DriveHandle] = -- INLINE
  BEGIN OPEN channelObj: base[LOOPHOLE[channel, ChannelHandle]];
  RETURN[channelObj.driveHandle]
  END;

NotifyIOComplete: PUBLIC ENTRY PROC [req: DiskChannel.IORequestHandle] =
  -- used by driver to indicate some IO transaction has completed.
  BEGIN
  OPEN channelObj: base[LOOPHOLE[req.channel, ChannelHandle]],
    channelObj.completionHandle;
  channelObj.ioCount ← channelObj.ioCount - 1;
  req.next ← NIL;
  IF first = NIL THEN first ← req ELSE last.next ← req;
  last ← req;
  BROADCAST ioComplete;
  END;

RegisterDrive: PUBLIC ENTRY PROC [drive: DiskChannelBackend.DriveHandle] =
  -- Add new drive to end of list, to preserve enumeration order defined by faces
  BEGIN
  prevDrive: DiskChannelBackend.DriveHandle;
  drive.driveOrdinal ← 0; -- in case first of type
  drive.next ← NIL;
  drive.stateChangeLocked ← FALSE;
  IF g.firstDrive = NIL THEN g.firstDrive ← drive
  ELSE
    BEGIN
    FOR prevDrive ← g.firstDrive, prevDrive.next WHILE prevDrive.next ~= NIL DO
      ENDLOOP;
    prevDrive.next ← drive;
    IF prevDrive.driveID.type = drive.driveID.type THEN
      drive.driveOrdinal ← prevDrive.driveOrdinal + 1;
```



```
    END;
  END;

-- Initialization

g.firstDrive ← NIL;
Process.SetTimeout[@ioCompletePrototype, Process.MsecToTicks[1000]];
Process.InitializeCondition[@g.channelRestarted, Process.MsecToTicks[1000]];
Process.InitializeCondition[@shortTime, Process.MsecToTicks[5000]];
Process.InitializeCondition[@driveObjectUnlocked, Process.MsecToTicks[30000]];
END.

LOG
Time: January 11, 1979 3:28 PM By: Horsley Action: Create file
Time: August 14, 1979 11:25 PM By: Redell Action: New DiskChannel
Time: August 16, 1979 11:04 PM By: Redell Action: Add drive tags
Time: October 19, 1979 9:11 AM By: McJones Action: Avoid calling InitializeCondition in Create
**CompletionObject
Time: November 29, 1979 3:21 PM By: Gobbel Action: New DiskChannel interface: initialize cou
**ntDone field of IORequest
Time: January 31, 1980 4:53 PM By: McJones Action: New DiskChannel; preserve order of drive
**registration
Time: June 11, 1980 3:24 PM By: Luniewski Action: Changes for DiskChannel.Drive's being LO
**NG POINTER's to DriveObject's. Implement AwaitStateChange, SetDriveState. Make changes t
**o the channel implementation to account for removable volumes.
Time: June 23, 1980 4:16 PM By: McJones Action: OISDisk = >PilotDisk
Time: July 19, 1980 3:09 PM By: Jose Action: Changes to SetDriveState to fix deadlock, handle
**change of state.
Time: July 30, 1980 2:15 PM By: Luniewski Action: Modify SetDriveState to return status.
Time: August 7, 1980 9:27 AM By: Luniewski Action: Add support for stateChangeLock in drive
**Object.
Time: September 16, 1980 11:59 PM By: Jose Action: Get ready before changeCount in GetDriv
**eAttributes, change stateChangeLock to stateChangeLocked.
Time: October 3, 1980 12:07 PM By: Luniewski Action: AwaitStateChange: access all drives
```

```
-- DiskDriverSharedImpl.mesa (last edited by: McJones on: February 1, 1980 9:22 AM)

DIRECTORY
  DiskChannel: FROM "DiskChannel" USING [
    DiskPageNumber, IORequestHandle, Cylinder],
  DiskChannelBackend: FROM "DiskChannelBackend" USING [GetDrive],
  DiskDriverShared: FROM "DiskDriverShared" USING [ScheduleHandle];

DiskDriverSharedImpl: PROGRAM
  IMPORTS DiskChannelBackend EXPORTS DiskDriverShared SHARES DiskChannel =
  BEGIN
  -- Useful permanent statisitics
  totalNumberOfRequests: LONG CARDINAL ← 0;
  countQueueEmptyWhenRequestArrived: LONG CARDINAL ← 0;
  countSequentialRequestsOnSameCylinder: LONG CARDINAL ← 0;
  countOtherRequestsOnCurrentCylinder: LONG CARDINAL ← 0;
  countRequestsNotOnCurrentCylinder: LONG CARDINAL ← 0;
  countQueueEmptyWhenRequestMade: LONG CARDINAL ← 0;
  GetSchedule: PROCEDURE [req: DiskChannel.IORequestHandle]
    RETURNS [DiskDriverShared.ScheduleHandle] =
    -- Fancy loophole
    INLINE BEGIN RETURN[LOOPHOLE[DiskChannelBackend.GetDrive[req.channel]]]; END;

  GetNextPendingRequest: PUBLIC PROCEDURE [
    schedule: DiskDriverShared.ScheduleHandle]
    RETURNS [req: DiskChannel.IORequestHandle] =
    -- Pick next request off of queue
    BEGIN
    IF (req ← schedule.first) = NIL THEN
      BEGIN
      countQueueEmptyWhenRequestMade ← countQueueEmptyWhenRequestMade + 1;
      IF collectStats THEN RememberRequest[remove, 0];
      RETURN;
      END;
    schedule.first ← req.next;
    IF schedule.first = NIL OR req = schedule.previousRequest THEN
      schedule.previousRequest ← NIL;
    -- makes sure that previousRequest isn't pointing to something that's been removed from the q
**ueue
    totalNumberOfRequests ← totalNumberOfRequests + 1;
    IF collectStats THEN RememberRequest[remove, req.diskPage];
    END;

  InsertRequest: PUBLIC PROCEDURE [req: DiskChannel.IORequestHandle] =
    -- using a somewhat odd implementation of the elevator algorithm
    BEGIN
    InsertBefore: PROCEDURE [
      action: {firstLarger, firstSmaller},
      startingPlaceForInsertion: LONG POINTER TO DiskChannel.IORequestHandle] =
      -- find the correct place in the queue for the request
      INLINE
      BEGIN
      last: LONG POINTER TO DiskChannel.IORequestHandle ←
        startingPlaceForInsertion;
      current: DiskChannel.IORequestHandle;
      DO
        current ← last↑;
        IF current = NIL OR req.address.cylinder = current.address.cylinder THEN
          BEGIN InsertWithinCylinder[last]; RETURN END;
        IF
          (action = firstSmaller AND req.address.cylinder >
            current.address.cylinder) OR
          (action = firstLarger AND req.address.cylinder <
            current.address.cylinder) THEN
          BEGIN last↑ ← req; req.next ← current; RETURN END;
        last ← @current.next;
        ENDLOOP;
      END;
    InsertWithinCylinder: PROCEDURE [
      startingPlaceForInsertion: LONG POINTER TO DiskChannel.IORequestHandle] =
      -- linear insert into correct position amongst entries on current cylinder
      BEGIN
      last: LONG POINTER TO DiskChannel.IORequestHandle ←
        startingPlaceForInsertion;
      current: DiskChannel.IORequestHandle;
      DO
        current ← last↑;
        IF current = NIL OR req.address.cylinder ~= current.address.cylinder OR
          req.address.head < current.address.head OR
          (req.address.head = current.address.head AND req.address.sector <
            current.address.sector) THEN
          BEGIN last↑ ← req; req.next ← current; RETURN END;
        last ← @current.next;
        ENDLOOP;
      END;
    schedule: DiskDriverShared.ScheduleHandle = GetSchedule[req];
    currentCylinderOfDrive: DiskChannel.Cylinder = schedule.currentCylinder;
    IF collectStats THEN RememberRequest[insert, req.diskPage];
    req.next ← NIL;
    SELECT TRUE FROM
      schedule.first = NIL =>
        -- this case is also caught below, but it's useful to keep this statistic independently
        BEGIN
        schedule.first ← req;
        countQueueEmptyWhenRequestArrived ← countQueueEmptyWhenRequestArrived + 1;
        END;
      -- catch the typical case of sequential requests

      schedule.previousRequest ~= NIL AND req.address.cylinder =
        schedule.previousRequest.address.cylinder AND req.diskPage =
        schedule.previousRequest.diskPage + 1 =>
        BEGIN
        InsertWithinCylinder[@schedule.previousRequest.next];
        countSequentialRequestsOnSameCylinder ←
          countSequentialRequestsOnSameCylinder + 1;
        END;
      req.address.cylinder = currentCylinderOfDrive =>
        -- this case happens so seldom, it could probably be removed (allowing InsertWithinCylinde
**r to become an INLINE)
        BEGIN
        InsertWithinCylinder[@schedule.first];
        countOtherRequestsOnCurrentCylinder ←
          countOtherRequestsOnCurrentCylinder + 1;
        END;
      ENDCASE =>
        -- insert before the first queue entry with a smaller (or larger) cylinder value (merge with ent
**ries with equal cylinder value)
        BEGIN
```

```
        InsertBefore[
          IF req.address.cylinder > currentCylinderOfDrive THEN firstLarger
          ELSE firstSmaller, @schedule.first];
        countRequestsNotOnCurrentCylinder ← countRequestsNotOnCurrentCylinder + 1;
        END;
    schedule.previousRequest ← req;
    END;

  PeekNextPendingRequest: PUBLIC PROCEDURE [
    schedule: DiskDriverShared.ScheduleHandle]
    RETURNS [req: DiskChannel.IORequestHandle] =
    -- Pick next request without removing it from queue
    BEGIN RETURN[schedule.first] END;
  -- Temporary statistics gathering stuff: should be removed eventually

  collectStats: BOOLEAN = TRUE;
  RequestDirection: TYPE = {insert, remove};
  maxStatsRequests: CARDINAL = 64;
  currentIndex: CARDINAL ← 0;
  totalReqs: CARDINAL ← 0;
  pgs: ARRAY [0..maxStatsRequests) OF DiskChannel.DiskPageNumber;
  dirs: ARRAY [0..maxStatsRequests) OF RequestDirection;
  RememberRequest: PROCEDURE [
    dir: RequestDirection, pNum: DiskChannel.DiskPageNumber] =
    BEGIN
    IF currentIndex >= maxStatsRequests THEN currentIndex ← 0;
    pgs[currentIndex] ← pNum;
    dirs[currentIndex] ← dir;
    currentIndex ← currentIndex + 1;
    totalReqs ← totalReqs + 1;
    END;

  END.
LOG
Time: January 17, 1979 4:40 PM          By: Horsley          Action: Create file
Time: March 13, 1979 6:28 PM  By: Horsley          Action: RememberRequest no longer reme
**mbers physical address
Time: July 24, 1979 2:23 PM     By: Gobbel          Action: Change format of IORequest
Time: August 17, 1979 9:39 AM By: Redell Action: Add USINGs; remove use of old Temporary inte
**rface
Time: February 1, 1980 9:22 AM          By: McJones          Action: Changes to DiskChannel,
**DiskChannelBackend, DiskDriverShared
```

```
-- FloppyChannelImpl.mesa (last edited by: Jose on: October 21, 1980 2:32 PM)
-- THINGS TO DO:
-- 1) Protect against multiple simultaneous calls
-- 2) Troy format

DIRECTORY
  DeviceTypes USING [sa800],
  DiskChannel USING [PVHandle],
  DiskChannelBackend USING [
    ChangeStateProc, DriveHandle, DriveObject, GetStatusProc, RegisterDrive],
  DiskDriverShared USING [ScheduleHandle, ScheduleObject],
  DriverStartChain USING [Start],
  Environment USING [Base, first64K, wordsPerPage],
  FloppyChannel USING [Attributes, Buffer, Context, DiskAddress, Status],
  FloppyChannelInternal USING [
    maxDrives, maxSectorsPerTrack, maxWordsPerSector, OpBlock],
  Inline USING [BITAND, LongCOPY],
  PhysicalVolume USING [Error, ErrorType],
  PilotFloppyFormat USING [firstRealCylinder, logicalTrack0, pilotContext],
  Process USING [MsecToTicks, SetTimeout],
  ProcessInternal USING [AllocateNakedCondition],
  ResidentHeap USING [MakeNode],
  SA800Face USING [
    Attributes, Buffer, Context, DeviceHandle, DiskAddress, DiskChangeClear,
    Function, GetContext, GetDeviceAttributes, GetNextDevice,
    initialAllocationLength, Initialize, InitializeCleanup, Initiate,
    nullDeviceHandle, operationBlockLength, OperationPtr, Poll, SetContext,
    Status],
  Space USING [Create, Handle, LongPointer, Map, virtualMemory],
  SpecialSpace USING [MakeCodeResident, MakeResident, MakeSwappable],
  Transaction USING [];

FloppyChannelImpl: MONITOR
  IMPORTS
    DiskChannelBackend, RemainingDrivers: DriverStartChain, Inline,
    PhysicalVolume, Process, ProcessInternal, ResidentHeap, SA800Face, Space,
    SpecialSpace, Transaction
  EXPORTS
    DiskChannel, DriverStartChain, FloppyChannel, FloppyChannelInternal,
    PhysicalVolume =
  BEGIN OPEN FloppyChannel, FloppyChannelInternal;

  -- Constants

  maxDrives: CARDINAL = FloppyChannelInternal.maxDrives;
  maxRetries: CARDINAL = 30;
  maxSectorsPerTrack: CARDINAL = FloppyChannelInternal.maxSectorsPerTrack;
  maxWordsPerSector: CARDINAL = FloppyChannelInternal.maxWordsPerSector;
  nTasks: CARDINAL = 2;

  -- Types

  DriveObject: PUBLIC TYPE = DiskChannelBackend.DriveObject; --exported type
  --PhysicalVolume.--
  Handle: PUBLIC TYPE = DiskChannel.PVHandle; --exported type

  -- Global Variables
```



```
  clientDataSpace: Space.Handle;
  drives: ARRAY [0..maxDrives) OF DiskDriverShared.ScheduleObject;
  nakedNotify: LONG POINTER TO CONDITION;
  nDrives: CARDINAL ← 0;
  tasks: ARRAY [0..nTasks) OF LONG POINTER TO UNSPECIFIED;
  wakeupMask: WORD;

  -- EXTERNAL PROCEDURES

  ReadSectors: PUBLIC PROCEDURE [
    handle: Handle, address: DiskAddress, buffer: Buffer, count: CARDINAL,
    incrementDataPtr: BOOLEAN] RETURNS [status: Status, countDone: CARDINAL] =
    BEGIN
    op: OpBlock;
    AssertNonPilotAccess[handle];
    op ←
      [handle, readSector, GetDA[address], GetBuffer[buffer], incrementDataPtr,
        count];
    [status, countDone] ← DoClientRequest[@op];
    END;

  WriteSectors: PUBLIC PROCEDURE [
    handle: Handle, address: DiskAddress, buffer: Buffer, count: CARDINAL,
    incrementDataPtr: BOOLEAN] RETURNS [status: Status, countDone: CARDINAL] =
    BEGIN
    op: OpBlock ←
      [handle, writeSector, GetDA[address], GetBuffer[buffer], incrementDataPtr,
        count];
    [status, countDone] ← WriteInternal[@op];
    END;

  WriteDeletedSectors: PUBLIC PROCEDURE [
    handle: Handle, address: DiskAddress, buffer: Buffer, count: CARDINAL,
    incrementDataPtr: BOOLEAN] RETURNS [status: Status, countDone: CARDINAL] =
    BEGIN
    op: OpBlock ←
      [handle, writeDeletedSector, GetDA[address], GetBuffer[buffer],
        incrementDataPtr, count];
    [status, countDone] ← WriteInternal[@op];
    END;

  ReadID: PUBLIC PROCEDURE [handle: Handle, address: DiskAddress, buffer: Buffer]
    RETURNS [status: Status] =
    BEGIN
    op: OpBlock ← [handle, readID, GetDA[address], GetBuffer[buffer]];
    -- could check buffer length >= idLength
    AssertNonPilotAccess[handle];
    RETURN[DoRequest[@op].status];
    END;

  Nop: PUBLIC PROCEDURE [handle: Handle] RETURNS [status: Status] =
    BEGIN
    -- Nop can be called anytime, regardless of DiskChannel.DriveState, if a floppy drive is present
    op: OpBlock ← [handle, nop];
    RETURN[DoRequest[@op].status];
    END;

  Recalibrate: PUBLIC PROCEDURE [handle: Handle] RETURNS [status: Status] =
    BEGIN
```

```
    op: OpBlock ← [handle, recalibrate];
    AssertNonPilotAccess[handle];
    RETURN[DoRequest[@op].status];
    END;

  GetContext: PUBLIC PROCEDURE [handle: Handle] RETURNS [context: Context] =
    BEGIN RETURN[LOOPHOLE[SA800Face.GetContext[GetDeviceHandle[handle]]]]; END;

  GetDeviceAttributes: PUBLIC PROCEDURE [handle: Handle]
    RETURNS [attributes: Attributes] =
    BEGIN
    RETURN[LOOPHOLE[SA800Face.GetDeviceAttributes[GetDeviceHandle[handle]]]];
    END;

  SetContext: PUBLIC PROCEDURE [handle: Handle, context: Context]
    RETURNS [ok: BOOLEAN] =
    BEGIN
    ok ← SA800Face.SetContext[
      GetDeviceHandle[handle], LOOPHOLE[context, SA800Face.Context]];
    InitDisk[handle.drive, context];
    END;

  -- PROCEDURES EXPORTED VIA DRIVE OBJECT

  BasicChangeState: DiskChannelBackend.ChangeStateProc =
    -- Removable volume state change handler for state change independent of Pilot/notPilot stat
**e.
    -- This procedure is superceded by PilotChangeState when the Pilot driver world is established
**.
    BEGIN END;

  GetStatus: DiskChannelBackend.GetStatusProc =
    -- get online/offline status of drive
    BEGIN
    ready ← (NOT Nop[[LOOPHOLE[dH], dH.changeCount]].status.notReady);
    RETURN[ready, dH.changeCount];
    END;

  -- ENTRY PROCEDURES

  DoRequest: PUBLIC ENTRY PROCEDURE [pOp: POINTER TO OpBlock]
    RETURNS [status: FloppyChannel.Status, countDone: CARDINAL] =
    BEGIN -- Perform transfer requests, doing error recovery and retry.
    -- Access validity is checked further up the call chain:
    --  in DiskChannel.InitiateIO for Pilot volume; in FloppyChannel procedures for channel access
**.
    retryCount: CARDINAL ← 0;
    req: SA800Face.OperationPtr = tasks[0];
    drive: DiskChannelBackend.DriveHandle = LOOPHOLE[pOp.device.drive];

    FatalError: INTERNAL PROCEDURE RETURNS [fatal: BOOLEAN] =
      BEGIN
      oneThird: CARDINAL = maxRetries/3;
      SELECT TRUE FROM
        status.wrongSizeBuffer, status.recordNotFound, status.crcError,
          status.hardwareError =>
          BEGIN -- retryable error
          IF (retryCount ← retryCount + 1) > maxRetries THEN RETURN[TRUE];
```



```
          IF retryCount MOD oneThird = 0 THEN
            IF Recover[
              pOp.device,
              IF retryCount = oneThird THEN recover ELSE recal].status.error THEN
              RETURN[TRUE];
          IF FilterStatus[SA800Face.Initiate[req].status].error THEN RETURN[TRUE];
          RETURN[FALSE];
          END;
        ENDCASE => --fatal error--RETURN[TRUE];
      END;

    req↑ ←
      [device: drive.driveID.handle, function: pOp.function,
        incrementDataPointer: pOp.incrementDataPtr, address: pOp.address,
        buffer: pOp.buffer, count: pOp.count];
    [status: status] ← FilterStatus[SA800Face.Initiate[req].status];
    IF status.inProgress THEN
      DO
        -- wait for completion
        status ← FilterStatus[SA800Face.Poll[req]];
        SELECT TRUE FROM
          status.inProgress => WAIT nakedNotify;
          status.goodCompletion => EXIT;
          ENDCASE => --error--IF FatalError[] THEN EXIT;
        ENDLOOP;

    countDone ← pOp.count - req.count;
    IF status.diskChanged THEN NoticeDiskChanged[drive];

    END;

  InitDisk: ENTRY PROCEDURE [
    drive: DiskChannelBackend.DriveHandle, context: FloppyChannel.Context] =
    BEGIN
    -- Set physical volume(diskette)-specific information when context changes
    nCyl: CARDINAL ←
      SA800Face.GetDeviceAttributes[
        drive.driveID.handle].attributes.numberOfCylinders -
        PilotFloppyFormat.logicalTrack0 - PilotFloppyFormat.firstRealCylinder;
    i: CARDINAL;
    sectorTable: ARRAY [0..5] OF CARDINAL = [4, 8, 15, 26, 36, 2];
    i ←
      SELECT context.sectorLength FROM
        512 => 1, -- 1024-byte sectors --
        256 => 2, -- 512 (Pilot volume) --
        128 => 3, -- 256 --
        64 => 4, -- 128 --
        ENDCASE => 5; -- Anything else must be Troy --
    IF context.density = single THEN i ← i - 1;
    drive.sectorsPerTrack ← sectorTable[i];
    drive.nPages ← drive.movingHeads*drive.sectorsPerTrack*nCyl;
    END;

  Recal: PUBLIC ENTRY PROCEDURE [handle: Handle]
    RETURNS [status: FloppyChannel.Status] =
    BEGIN RETURN[Recover[handle, recal]] END;

  -- INTERNAL PROCEDURES
```

```
  FilterStatus: INTERNAL PROCEDURE [s: SA800Face.Status]
    RETURNS [status: FloppyChannel.Status] =
    -- Convert Face-type status to FloppyChannel variety
    BEGIN
    initialMask: SA800Face.Status =
      [ --Face status elements to keep--diskChange: TRUE, na1: FALSE,
        twoSided: TRUE, na3: FALSE, error: TRUE, inProgress: TRUE,
        recalibrateError: TRUE, dataLost: TRUE, notReady: TRUE,
        writeProtect: TRUE, deletedData: TRUE, recordNotFound: TRUE,
        crcError: TRUE, track00: TRUE, index: FALSE, busy: FALSE];
    status ← LOOPHOLE[Inline.BITAND[s, initialMask]];
    SELECT TRUE FROM
      s.inProgress, s.error => NULL;
      s.na1, s.na3 => BEGIN status.hardwareError ← TRUE; status.error ← TRUE END;
      ENDCASE => status.goodCompletion ← TRUE;
    END;

  NoticeDiskChanged: INTERNAL PROCEDURE [drive: DiskChannelBackend.DriveHandle] =
    BEGIN
    drive.changeCount ← drive.changeCount + 1;
    SA800Face.DiskChangeClear[drive.driveID.handle];
    END;

  Recover: INTERNAL PROCEDURE [handle: Handle, func: {recover, recal}]
    RETURNS [status: FloppyChannel.Status] =
    -- Perform recover/recalibrate outside normal request path. Polls until complete; error recove
**ry = simple retry.
    BEGIN
    drive: DiskChannelBackend.DriveHandle ← LOOPHOLE[handle.drive];
    req: SA800Face.OperationPtr = tasks[1];
    retryLimit: CARDINAL = 10;
    req↑ ←
      [device: drive.driveID.handle,
        function: IF func = recal THEN recalibrate ELSE recovery,
        incrementDataPointer: FALSE, address: [0, 0, 0], buffer: [NIL, 0],
        count: 0];
    THROUGH [0..retryLimit) DO
      status ← FilterStatus[SA800Face.Initiate[req].status];
      IF status.error THEN EXIT;
      DO
        -- wait for completion
        status ← FilterStatus[SA800Face.Poll[req]];
        IF NOT status.inProgress THEN EXIT;
        WAIT nakedNotify;
        ENDLOOP;
      IF status.goodCompletion THEN RETURN;
      ENDLOOP; -- here only on error
    IF status.diskChanged THEN NoticeDiskChanged[drive];
    END;

  -- ETC.

  AssertChannelAccess: PROCEDURE [handle: Handle] =
    -- Ensure the floppy is open for channel access only - must not be Pilot or inactive.
    BEGIN
    drive: DiskChannelBackend.DriveHandle ← GetDrive[handle];
    IF (drive.changeCount # handle.changeCount) OR (drive.state # channel) THEN
      PhysicalVolume.Error[invalidHandle];
    END;
```



```
  AssertNonPilotAccess: PROCEDURE [handle: Handle] =
    -- Ensure the floppy is not open for Pilot access - may be channel or inactive.
    BEGIN
    drive: DiskChannelBackend.DriveHandle ← GetDrive[handle];
    IF (drive.changeCount # handle.changeCount) OR (drive.state = pilot) THEN
      PhysicalVolume.Error[invalidHandle];
    END;

  DoClientRequest: PROCEDURE [pOp: POINTER TO OpBlock]
    RETURNS [status: FloppyChannel.Status, countDone: CARDINAL] =
    -- Intermediate stop on the way to DoRequest which takes care of pinning client's data for
    -- ReadSectors, ReadID, WriteSectors, and WriteDeletedSectors. Transfers are done in appro
**ximately
    -- track-length chunks.
    BEGIN
    -- remove commenting on next statement when DoRequest clients use new buffer.length sema
**ntics
    bufferLength: CARDINAL =
      maxSectorsPerTrack*GetContext[pOp.device].context.sectorLength;
    clientReq: OpBlock ← pOp↑; -- does this really need to be copied?
    countDoneThisPass: CARDINAL;
    countLeft: CARDINAL ← pOp.count;
    func: {read, write} =
      SELECT pOp.function FROM readSector, readID => read, ENDCASE => write;
    pPinned: LONG POINTER ← Space.LongPointer[clientDataSpace];
    pUser: LONG POINTER ← pOp.buffer.address;
    userLength: CARDINAL ← pOp.buffer.length;
    SpecialSpace.MakeResident[clientDataSpace];
    IF pUser ~= NIL THEN clientReq.buffer.address ← pPinned;
    clientReq.incrementDataPtr ← FALSE;
    WHILE countLeft > 0 DO
      length: CARDINAL;
      clientReq.count ← MIN[countLeft, maxSectorsPerTrack];
      --would be nice to do only current track
      clientReq.buffer.length ← length ← MIN[userLength, bufferLength];
      IF func = write THEN
        Inline.LongCOPY[from: pUser, nwords: length, to: pPinned];
      [status, countDoneThisPass] ← DoRequest[@clientReq];
      IF func = read THEN
        Inline.LongCOPY[from: pPinned, nwords: length, to: pUser];
      countLeft ← countLeft - countDoneThisPass;
      IF status.error THEN EXIT;
      IF pOp.incrementDataPtr THEN {
        pUser ← pUser + length; userLength ← userLength - length};
      ENDLOOP;
    countDone ← pOp.count - countLeft;
    SpecialSpace.MakeSwappable[clientDataSpace];
    END;

  GetBuffer: PROCEDURE [buf: FloppyChannel.Buffer]
    RETURNS [buffer: SA800Face.Buffer] =
    -- Make a Face-style Buffer address from a Floppy Channel one.
    BEGIN -- should check buffer length for equal to sector size
    RETURN[SA800Face.Buffer[address: buf.address, length: buf.length]];
    END;

  GetDA: PROCEDURE [addr: FloppyChannel.DiskAddress]
    RETURNS [address: SA800Face.DiskAddress] =
```

```
    -- Make a Disk address acceptable to the Face from the one given the user.
    BEGIN
    RETURN[
      SA800Face.DiskAddress[
        cylinder: addr.cylinder, head: addr.head, sector: addr.sector]];
    END;

  GetDeviceHandle: PROCEDURE [handle: Handle]
    RETURNS [device: SA800Face.DeviceHandle] =
    -- Extract the device handle expected by the Face from the drive object pointed to by the client'
**s handle.
    BEGIN RETURN[GetDrive[handle].driveID.handle]; END;

  GetDrive: PROCEDURE [handle: Handle]
    RETURNS [device: DiskChannelBackend.DriveHandle] =
    -- Thread a path to the drive object.
    BEGIN RETURN[LOOPHOLE[handle.drive]]; END;

  WriteInternal: PROCEDURE [pOp: POINTER TO OpBlock]
    RETURNS [status: Status, countDone: CARDINAL] =
    -- Write(Sectors/DeletedSectors) and read after write.
    BEGIN
    readStatus: Status;
    AssertChannelAccess[pOp.device];
    [status, countDone] ← DoClientRequest[pOp];
    IF countDone > 0 THEN -- read after (some successful) write
      BEGIN
      pOp.count ← countDone;
      pOp.function ← readSector;
      [readStatus, countDone] ← DoClientRequest[pOp];
      IF countDone < pOp.count THEN status ← readStatus;
      -- Read status returned only if read-after-write fails earlier than write.
      -- In this case an unknown number of sectors beyond the read failure may
      -- have been written.

      END;
    END;

  -- INITIALIZATION PROCEDURES

  InitializeCondition: PROCEDURE =
    BEGIN
    [cv: nakedNotify, mask: wakeupMask] ← ProcessInternal.AllocateNakedCondition[
      ];
    Process.SetTimeout[condition: nakedNotify, ticks: Process.MsecToTicks[1000]];
    END;

  InitializeDriverAlloc: PROCEDURE = INLINE
    BEGIN
    opBase: Environment.Base RELATIVE POINTER;
    trackPages: CARDINAL =
      (maxSectorsPerTrack*maxWordsPerSector + Environment.wordsPerPage -
        1)/Environment.wordsPerPage;
    [node: opBase] ← ResidentHeap.MakeNode[
      n: SA800Face.operationBlockLength*2, alignment: a4];
    tasks[0] ← @Environment.first64K[opBase];
    FOR i: CARDINAL IN [1..nTasks) DO
      tasks[i] ← tasks[i - 1] + SA800Face.operationBlockLength; ENDLOOP;
    clientDataSpace ← Space.Create[parent: Space.virtualMemory, size: trackPages];
```



```
    -- This is ~4K words
    Space.Map[clientDataSpace];
    END;

  RegisterDrives: PROCEDURE = -- Create and register drive objects.
    BEGIN
    attr: SA800Face.Attributes;
    device: SA800Face.DeviceHandle ← SA800Face.nullDeviceHandle;
    d: DiskDriverShared.ScheduleHandle;
    gsp: Environment.Base RELATIVE POINTER;
    [node: gsp] ← ResidentHeap.MakeNode[
      n: SA800Face.initialAllocationLength, alignment: a16];
    SA800Face.Initialize[
      notify: wakeupMask, initialAllocation: @Environment.first64K[gsp]];
    WHILE nDrives < maxDrives AND
      ((device ← SA800Face.GetNextDevice[device]) ~= SA800Face.nullDeviceHandle)
      DO
      SA800Face.DiskChangeClear[device];
      d ← @drives[nDrives];
      d.drive.driveID ← [type: DeviceTypes.sa800, handle: device];
      attr ← SA800Face.GetDeviceAttributes[device];
      d.drive.cylinders ← attr.numberOfCylinders;
      d.drive.movingHeads ← attr.numberOfHeads;
      d.drive.sectorsPerTrack ← 0;
      d.drive.nPages ← 0;
      d.drive.driverStorage ← NIL;
      d.drive.state ← inactive;
      d.drive.changeCount ← 0;
      d.drive.requestIO ← NIL;
      d.drive.getPageAddress ← NIL;
      d.drive.getPageNumber ← NIL;
      d.drive.getStatus ← GetStatus;
      d.drive.changeState ← BasicChangeState;
      d.currentSector ← 0;
      d.currentCylinder ← 0;
      d.first ← NIL;
      DiskChannelBackend.RegisterDrive[@d.drive];
      [] ← SetContext[
        [@d.drive, d.drive.changeCount], PilotFloppyFormat.pilotContext];
      nDrives ← nDrives + 1;
      ENDLOOP;
    IF nDrives ~= 0 THEN
      BEGIN
      SA800Face.InitializeCleanup[device];
      InitializeDriverAlloc[];
      SpecialSpace.MakeCodeResident[FloppyChannelImpl];
      END;
    END;

  Start: PUBLIC PROCEDURE = -- exported to (StoreDriver)StartChain
    BEGIN RemainingDrivers.Start[]; END; -- INITIALIZATION

  InitializeCondition[];
  RegisterDrives[];

  END.

May 28, 1980 4:26 PM  Jose  Create file
```

August 28, 1980 12:20 PM McJones Convert to ProcessInternal.AllocateNakedCondition
September 18, 1980 4:23 PM Jose Separate startup- and new diskette-initialization; pin co
***de and allocate driver memory only if drive present, incorporate new SA800Face*
October 10, 1980 4:21 PM Jose Rip out runs of pages code, substitute head runs, add
***DoClientRequest.*
October 21, 1980 2:32 PM Jose Return earliest failure status on Write(Deleted)Sectors,
***catch initiate error in DoRequest.*

```
-- SA4000Impl.mesa (last edited by: McJones on: August 29, 1980 5:00 PM)
-- THINGS TO DO:
-- 1) Allocate ScheduleObject's dynamically

DIRECTORY
  DeviceTypes USING [sa4000],
  DiskChannel USING [
    CompletionStatus, Cylinder, DiskPageNumber, DriveState, IORequestHandle],
  DiskChannelBackend USING [
    ChangeStateProc, DriveHandle, GetDrive, GetPAProc, GetPNProc, GetStatusProc,
    NotifyIOComplete, RegisterDrive, RequestIOProc],
  DiskDriverShared USING [
    ScheduleHandle, ScheduleObject, GetNextPendingRequest, InsertRequest,
    PeekNextPendingRequest],
  Environment USING [Base, first64K],
  Inline USING [LongDivMod, LowHalf],
  Process USING [Detach, MsecToTicks, SetPriority, SetTimeout],
  ProcessInternal USING [AllocateNakedCondition],
  ProcessPriorities USING [diskInterruptPriority],
  ResidentHeap USING [MakeNode],
  RuntimeInternal USING [WorryCallDebugger],
  SA4000Face USING [
    Command, DeviceHandle, DiskAddress, GetDeviceAttributes, GetNextDevice,
    GlobalStatePtr, globalStateSize, Initialize, InitializeCleanup, Initiate,
    nullDeviceHandle, Operation, OperationPtr, operationSize, Poll, Recalibrate,
    Status],
  StoreDriverStartChain USING [Start],
  Utilities USING [LongPointerFromPage],
  Zone USING [Status];

SA4000Impl: MONITOR
  IMPORTS
    DiskChannelBackend, DiskDriverShared, Inline, Process, ProcessInternal,
    ResidentHeap, RuntimeInternal, SA4000Face,
    RemainingDrivers: StoreDriverStartChain, Utilities
  EXPORTS StoreDriverStartChain
  SHARES DiskChannel =
  BEGIN

  ErrorHalt: PROC = {RuntimeInternal.WorryCallDebugger["Error in SA4000Impl"]};

  ErrorHalt1: PROC RETURNS [SA4000Face.Command] = LOOPHOLE[ErrorHalt];

  currentDrive: CARDINAL;
  drives: ARRAY [0..3] OF DiskDriverShared.ScheduleObject;
  nakedNotify: LONG POINTER TO CONDITION;
  wakeupMask: WORD;
  nDrives: CARDINAL ← 0;
  maxRetries: CARDINAL = 8;
  operationsPerDrive: CARDINAL = 4; -- make this be the right number some time
  base: Environment.Base = Environment.first64K;

  -- EXTERNAL PROCS

  GetDiskAddress: PROC [
    dH: DiskChannelBackend.DriveHandle, page: DiskChannel.DiskPageNumber]
    RETURNS [addr: SA4000Face.DiskAddress] = INLINE
    BEGIN OPEN addr;
```



```
    temp: CARDINAL;
    [quotient: temp, remainder: sector] ← Inline.LongDivMod[
      num: page, den: dH.sectorsPerTrack];
    head ← temp MOD dH.movingHeads;
    cylinder ← temp/dH.movingHeads;
    END;

  GetPageAddress: DiskChannelBackend.GetPAProc = {
    RETURN[LOOPHOLE[GetDiskAddress[dH, page]]]};

  GetPageNumber: DiskChannelBackend.GetPNProc =
    BEGIN OPEN LOOPHOLE[page, SA4000Face.DiskAddress];
    RETURN[sector + dH.sectorsPerTrack*LONG[head + dH.movingHeads*cylinder]]
    END;

  GetStatus: DiskChannelBackend.GetStatusProc =
    -- It is always ready since removable SA4000's are not supported.
    {RETURN[ready: TRUE, changeCount: dH.changeCount]};

  ChangeState: DiskChannelBackend.ChangeStateProc = { --no-op for the SA4000--};

  InterruptProcess: PROC =
    BEGIN
    Process.SetPriority[ProcessPriorities.diskInterruptPriority];
    -- Main loop of interrupt process, which cannot be inside the monitor
    -- since it calls NotifyIOComplete:
    DO DiskChannelBackend.NotifyIOComplete[HandleInterrupt[]] ENDLOOP;
    END;

  -- ENTRY PROCS

  -- Temporary logging for timing interrupt handling
  -- start: INTEGER ← Inline.LowHalf[System.GetClockPulses[]];
  -- end: INTEGER;
  -- finished, started: INTEGER;
  -- LogIndex: TYPE = [1..100];
  -- log: ARRAY LogIndex OF LogEntry ← ALL[[0,0,0]];
  -- LogEntry: TYPE = RECORD [duration, finished, started: INTEGER];
  -- LogPtr: TYPE = LONG POINTER TO LogEntry;
  -- first: LogPtr = @log[FIRST[LogIndex]];
  -- last: LogPtr = @log[LAST[LogIndex]];
  -- logEntry: LogPtr ← first;

  statusTable: ARRAY SA4000Face.Status OF DiskChannel.CompletionStatus =
    [ --impossible--, goodCompletion, checksumError, checksumError,
      labelDoesNotMatch, seekFailed, hardwareError, notReady, hardwareError,
      hardwareError, hardwareError, hardwareError];

  HandleInterrupt: ENTRY PROC RETURNS [DiskChannel.IORequestHandle] =
    --INLINE
    -- This routine is called repeatedly until all completed operations have
    -- been returned to main loop;
    -- it then feeds the disk and goes to sleep waiting for the next interrupt.
    BEGIN
    RestartDisk: PROC = INLINE
      BEGIN
      qNode ← firstBusy; -- restart all active operations
      DO
```

```
      SA4000Face.Initiate[qNode.operation];
      qNode ← qNode.next;
      IF qNode = firstBusy OR qNode = firstFree THEN EXIT;
      ENDLOOP;
    END;
  req: DiskChannel.IORequestHandle;
  qNode: QNodePtr; -- Get next request to complete
  -- Note that face aborts all operations initiated after erroneous one, until Poll
  DO
    IF ~QueueEmpty[] THEN
      BEGIN
      status: SA4000Face.Status = SA4000Face.Poll[firstBusy.operation];
      IF status ~= inProgress THEN
        BEGIN
        -- got one
        req ← firstBusy.req;
        -- first pending operation has completed, for better or worse...
        req.status ← statusTable[status];
        IF req.count ~= req.countDone + firstBusy.operation.pageCount THEN
          req.retryCount ← 0;
        -- zero retry count if we have completed some pages successfully
        req.countDone ← req.count - firstBusy.operation.pageCount;
        -- add number of pages we finished this time through
        IF status = goodCompletion THEN {FreeQNode[]; RETURN[req]};
        IF
          (status = labelCheck AND req.countDone # 0
            -- AND req.expectLabelCheck)
            -- tempory, until expectLabelCheckField is added to DiskChannel--
            -- note this is coupled with FilerTransferImpl.VerifyLabel hack --
            AND req.dontIncrement AND req.command = vvr) OR
          (req.retryCount ← req.retryCount + 1) > maxRetries THEN {
          FreeQNode[]; IF ~QueueEmpty[] THEN RestartDisk[]; RETURN[req]};
        -- There has been a (soft) error
        IF (req.retryCount MOD (maxRetries/2)) = 0 THEN
          SA4000Face.Recalibrate[firstBusy.operation.device];
        RestartDisk[];
        END;
      END;
    -- No successfully completed requests
    FeedDisk[];
    -- started ← FeedDisk[];
    -- end ← Inline.LowHalf[System.GetClockPulses[]];
    -- logEntry↑ ← [duration: end-start, finished: finished, started: started];
    -- logEntry ← IF logEntry = last THEN first ELSE logEntry + SIZE[LogEntry];
    WAIT nakedNotify; -- start ← Inline.LowHalf[System.GetClockPulses[]];
    -- finished ← 0;

    ENDLOOP;
  END;

RequestIO: ENTRY DiskChannelBackend.RequestIOProc =
  BEGIN
  req.address ← GetPageAddress[
    DiskChannelBackend.GetDrive[req.channel], req.diskPage];
  req.retryCount ← 0;
  DiskDriverShared.InsertRequest[req];
  -- add to queue of pending requests
  --[] ←--
  FeedDisk[];
```



```
  END;

-- INTERNAL PROCS

-- should only be called when one or more queue nodes are available
-- (i.e. QueueFull[] = FALSE)

DoRequest: INTERNAL PROC [req: DiskChannel.IORequestHandle] = -- INLINE
  BEGIN
  device: SA4000Face.DeviceHandle = DiskChannelBackend.GetDrive[
    req.channel].driveID.handle;
  qNode: QNodePtr = AllocateQNode[];
  qNode.operation↑ ← SA4000Face.Operation[
    clientHeader: LOOPHOLE[req.address],
    diskHeader: SA4000Face.DiskAddress[0, 0, 0]; labelPtr: req.label,
    incrementDataPtr: ~req.dontIncrement,
    command:
    SELECT req.command FROM
      vvr => vvr,
      vvw => vvw,
      vrr => vrr,
      vww => vww,
      ENDCASE => ErrorHalt1[], pageCount: Inline.LowHalf[req.count],
    device: device, deviceStatus:,
    dataPtr: Utilities.LongPointerFromPage[req.memoryPage]];
  SA4000Face.Initiate[qNode.operation];
  qNode.req ← req;
  END;

FeedDisk: INTERNAL PROC -- RETURNS [i: INTEGER] -- =
  -- keep the disk running, if possible
  --INLINE--
  BEGIN
  req: DiskChannel.IORequestHandle; -- i ← 0;
  UNTIL QueueFull[] OR (req ← GetNextPendingRequest[]) = NIL DO
    DoRequest[req]; -- i ← i + 1;

    ENDLOOP;
  END;

-- allows current drive to complete all requests on the current cylinder, then
-- round-robins through the drives (no provision is made for overlapped seek).

GetNextPendingRequest: INTERNAL PROC RETURNS [DiskChannel.IORequestHandle] =
  --INLINE--
  BEGIN
  req: DiskChannel.IORequestHandle;
  drive: DiskDriverShared.ScheduleHandle ← @drives[currentDrive];
  -- always give priority to sectors on the current cylinder of the current drive
  req ← DiskDriverShared.PeekNextPendingRequest[drive];
  IF req ~= NIL AND drive.currentCylinder = req.address.cylinder THEN
    RETURN[DiskDriverShared.GetNextPendingRequest[drive]];
  -- whenever a cylinder change is required, go to next drive
  -- (hoping that the cylinder change might be avoided)
  THROUGH [0..nDrives) DO
    currentDrive ← currentDrive + 1;
    IF currentDrive >= nDrives THEN currentDrive ← 0;
    drive ← @drives[currentDrive];
```

```
    req ← DiskDriverShared.GetNextPendingRequest[drive];
    IF req ~= NIL THEN RETURN[req];
    ENDLOOP;
  RETURN[NIL]
  END;

-- Queue node machinery

QNodePtr: TYPE = LONG POINTER TO QNode;
QNode: TYPE = RECORD [
  operation: SA4000Face.OperationPtr,
  req: DiskChannel.IORequestHandle,
  next: QNodePtr];
firstBusy, firstFree: QNodePtr;

AllocateQNode: INTERNAL PROC RETURNS [qNode: QNodePtr] = INLINE
  BEGIN
  qNode ← firstFree;
  IF (firstFree ← firstFree.next) = firstBusy THEN firstFree ← NIL
  END;

FreeQNode: INTERNAL PROC = INLINE {
  IF QueueFull[] THEN firstFree ← firstBusy; firstBusy ← firstBusy.next};

QueueFull: INTERNAL PROC RETURNS [BOOLEAN] = INLINE {RETURN[firstFree = NIL]};

QueueEmpty: INTERNAL PROC RETURNS [BOOLEAN] = INLINE {
  RETURN[firstFree = firstBusy]};

-- INITIALIZATION PROCS

RegisterDrives: PROC = -- Create and register drive objects.
  BEGIN
  device: SA4000Face.DeviceHandle ← SA4000Face.nullDeviceHandle;
  d: DiskDriverShared.ScheduleHandle;
  gsp: SA4000Face.GlobalStatePtr;
  status: Zone.Status;
  [gsp, status] ← ResidentHeap.MakeNode[
    n: SA4000Face.globalStateSize, alignment: a16];
  SA4000Face.Initialize[t: wakeupMask, globalState: gsp];
  WHILE (device ← SA4000Face.GetNextDevice[device]) ~=
    SA4000Face.nullDeviceHandle DO
    d ← @drives[nDrives];
    d.drive.driveID ← [type: DeviceTypes.sa4000, handle: device];
    [cylinders: d.drive.cylinders, movingHeads: d.drive.movingHeads,
      fixedHeads: d.drive.fixedHeads, sectorsPerTrack: d.drive.sectorsPerTrack]
      ← SA4000Face.GetDeviceAttributes[device];
    d.drive.nPages ←
      (d.drive.cylinders*d.drive.movingHeads +
        d.drive.fixedHeads)*d.drive.sectorsPerTrack;
    d.drive.requestIO ← RequestIO;
    d.drive.getPageAddress ← GetPageAddress;
    d.drive.getPageNumber ← GetPageNumber;
    d.drive.state ← inactive;
    d.drive.changeCount ← 1; -- removable SA4000's are not supported
    d.drive.getStatus ← GetStatus;
    d.drive.changeState ← ChangeState;
    d.currentSector ← 0;
```



```
    d.currentCylinder ← 0;
    d.first ← NIL;
    DiskChannelBackend.RegisterDrive[@d.drive];
    nDrives ← nDrives + 1;
    ENDLOOP;
  IF nDrives ~= 0 THEN SA4000Face.InitializeCleanup[];
  END;

InitGlobals: PROC =
  -- Set up ring of queue nodes and associated IOCB's.
  BEGIN
  MakeQNode: PROC RETURNS [qNode: QNodePtr] =
    BEGIN
    rp: Environment.Base RELATIVE POINTER TO QNode;
    status: Zone.Status;
    [rp, status] ← ResidentHeap.MakeNode[SIZE[QNode]];
    IF status ~= okay THEN ErrorHalt[];
    qNode ← @base[rp];
    END;
  MakeOperation: PROC RETURNS [operation: SA4000Face.OperationPtr] =
    BEGIN
    rp: Environment.Base RELATIVE POINTER TO SA4000Face.Operation;
    status: Zone.Status;
    [rp, status] ← ResidentHeap.MakeNode[
      n: SA4000Face.operationSize, alignment: a16];
    IF status ~= okay THEN ErrorHalt[];
    operation ← @base[rp];
    END;
  qNode, qNodeFirst: QNodePtr;
  qNode ← qNodeFirst ← MakeQNode[];
  qNode.operation ← MakeOperation[];
  THROUGH (0..operationsPerDrive*nDrives) DO
    qNode ← qNode.next ← MakeQNode[];
    qNode.operation ← MakeOperation[];
    ENDLOOP;
  qNode.next ← qNodeFirst; -- close the ring of queue nodes
  firstBusy ← firstFree ← qNode; -- initial state = queue empty

  END;

Start: PUBLIC PROC = {RemainingDrivers.Start[]};
-- exported to StoreDriverStartChain

-- Initialization

[cv: nakedNotify, mask: wakeupMask] ← ProcessInternal.AllocateNakedCondition[];
Process.SetTimeout[condition: nakedNotify, ticks: Process.MsecToTicks[1000]];
RegisterDrives[];
InitGlobals[];
Process.Detach[FORK InterruptProcess];
currentDrive ← 0; -- starting drive

END.
```

(For earlier log entries see Pilot 4.0 archive version.)
*April 13, 1980 10:35 PM Forrest Make IOCSImpl.InitializeNakedConditionVariable a loc**al inline*
June 11, 1980 10:08 AM Luniewski Initialize the state, changeCount and getStatus fields of

***DriveObject's correctly*
June 25, 1980 9:46 AM McJones Changes to SA4000Face for 10-word labels
July 19, 1980 3:12 PM Jose Changes for addition of changeState field to DriveObje
***ct*
July 24, 1980 2:52 PM Luniewski Initialize drive state in drive objects to inactive
August 28, 1980 11:56 AM McJones Convert to ProcessInternal.AllocateNakedCondition
October 11, 1980 9:01 PM Forrest hack for label check errors....

```
-- SA800Impl.mesa (last edited by: Jose on: October 21, 1980 2:15 PM)
-- THINGS TO DO:
-- 1) Allocate ScheduleObjects dynamically
-- 2) Handle multiple drives

DIRECTORY
  DeviceTypes USING [sa800],
  DiskChannel USING [
    Address, CompletionStatus, DiskPageNumber, Drive, GetNextDrive,
    IORequestHandle, Label, nullDrive, PVHandle],
  DiskChannelBackend USING [
    ChangeStateProc, DriveHandle, GetDrive, GetPAProc, GetPNProc,
    NotifyIOComplete, RequestIOProc],
  DriverStartChain USING [Start],
  Environment USING [wordsPerPage],
  FloppyChannel USING [Handle, SetContext, Status],
  FloppyChannelInternal USING [DoRequest, maxDrives, OpBlock],
  Inline USING [BITAND, DIVMOD, LongDiv, LowHalf],
  PhysicalVolume USING [],
  PilotDisk USING [MatchLabels, NextLabel],
  PilotFloppyFormat USING [
    firstRealCylinder, FloppyPageNumber, labelBasePage, logicalTrack0,
    pilotContext, wordsPerSector],
  Process USING [Detach, InitializeCondition, SecondsToTicks, SetPriority],
  ProcessPriorities USING [diskInterruptPriority],
  RuntimeInternal USING [WorryCallDebugger],
  SA800Face USING [DiskAddress, Function, GetDeviceAttributes],
  Space USING [
    Create, Delete, GetHandle, Handle, LongPointer, Map, nullHandle,
    PageFromLongPointer, PageNumber, PageOffset, virtualMemory],
  SpecialSpace USING [MakeCodeResident, MakeResident, MakeSwappable],
  Transaction USING [],
  Utilities USING [LongPointerFromPage];

SA800Impl: MONITOR
  IMPORTS
    DiskChannel, DiskChannelBackend, RemainingDrivers: DriverStartChain,
    FloppyChannel, FloppyChannelInternal, Inline, PilotDisk, Process,
    RuntimeInternal, SA800Face, Space, SpecialSpace, Transaction, Utilities
  EXPORTS DriverStartChain, PhysicalVolume
  SHARES DiskChannel =
  BEGIN
  -- Pilot Floppy Disk driver. This handles Pilot volumes on the floppy. Non-Pilot-volume access is
**accomplished via the interface FloppyChannel. The two forms of access are mutually interlock
**ed.
  --DiskChannel.--
  Drive: PUBLIC TYPE = DiskChannelBackend.DriveHandle; -- exported type
  --PhysicalVolume.--
  Handle: PUBLIC TYPE = DiskChannel.PVHandle; -- exported type
  ErrorHalt: PROCEDURE =
    BEGIN RuntimeInternal.WorryCallDebugger["Error in SA800Impl"] END;

  ErrorHalt1: PROCEDURE RETURNS [SA800Face.Function] = LOOPHOLE[ErrorHalt];
  maxDrives: CARDINAL = FloppyChannelInternal.maxDrives;
  nDrives: CARDINAL ← 0;
  requestSubmitted: CONDITION;
  -- Label machinery
  Labels: TYPE = ARRAY [0..0) OF DiskChannel.Label;
```



```
  PLabels: TYPE = LONG POINTER TO Labels;
  LabelObject: TYPE = RECORD [
    hasLabels: BOOLEAN ← FALSE, -- goes away when ChangeState returns error
    sH: Space.Handle ← Space.nullHandle,
    pLabels: PLabels ← NIL];
  labelBlock: ARRAY [0..maxDrives) OF LabelObject;
  labelSize: CARDINAL = SIZE[DiskChannel.Label];
  -- Request buffering machinery
  first, last: DiskChannel.IORequestHandle ← NIL;
  -- EXTERNAL PROCEDURES
  PilotChangeState: DiskChannelBackend.ChangeStateProc =
    -- Pilot-device device-dependent setup/cleanup when drive changes type of access from dH.st
**ate to state
    -- possible state changes: inactive -> pilot OR channel (or inactive) and vice versa
    BEGIN
    pLO: LONG POINTER TO LabelObject = dH.driverStorage;
    IF dH.state = pilot THEN
      BEGIN
      IF first ~= NIL THEN ERROR; -- request queue not empty
      Space.Delete[pLO.sH];
      END;
    SELECT state FROM -- new state

      pilot => InitLabelImage[dH];
      -- errors from label startup to be handled and turned into returned status

      channel =>
        dH.cylinders ← SA800Face.GetDeviceAttributes[
          dH.driveID.handle].attributes.numberOfCylinders;
      ENDCASE =>
        BEGIN --inactive--
        fHandle: DiskChannel.PVHandle = [LOOPHOLE[dH], dH.changeCount];
        [] ← FloppyChannel.SetContext[fHandle, PilotFloppyFormat.pilotContext];
        END;
    END;

  GetDiskAddress: PROCEDURE [
    dH: DiskChannelBackend.DriveHandle, page: PilotFloppyFormat.FloppyPageNumber]
    RETURNS [addr: SA800Face.DiskAddress] = INLINE
    BEGIN OPEN addr;
    temp: CARDINAL;
    [quotient: temp, remainder: sector] ← Inline.DIVMOD[
      num: page, den: dH.sectorsPerTrack];
    sector ← sector + 1; -- taking into account that floppy sectors are base 1
    head ← temp MOD dH.movingHeads;
    cylinder ← temp/dH.movingHeads + PilotFloppyFormat.firstRealCylinder;
    END;

  GetLogicalDiskAddress: PROCEDURE [
    dH: DiskChannelBackend.DriveHandle, page: PilotFloppyFormat.FloppyPageNumber]
    RETURNS [addr: DiskChannel.Address] =
    -- convert raw disk address to one relative to start of Pilot pages
    INLINE
    BEGIN OPEN addr;
    addr ← LOOPHOLE[GetDiskAddress[dH, page]];
    RETURN[
      [cylinder: cylinder + PilotFloppyFormat.logicalTrack0, head: head,
        sector: sector]]
```

```
  END;

GetPageAddress: DiskChannelBackend.GetPAProc =
  -- convert physical volume page number to physical disk address
  BEGIN RETURN[GetLogicalDiskAddress[dH, FloppyPage[page]]] END;

GetPageNumber: DiskChannelBackend.GetPNProc =
  -- convert physical disk address to physical volume page number
  BEGIN OPEN LOOPHOLE[page, SA800Face.DiskAddress];
  RETURN[
    (sector - 1) +
      dH.sectorsPerTrack*LONG[
        head + dH.movingHeads*(cylinder - PilotFloppyFormat.logicalTrack0)]]
  END;

RequestProcess: PROCEDURE =
  BEGIN
  countDone, labCountDone: CARDINAL;
  drive: DiskChannelBackend.DriveHandle;
  op: FloppyChannelInternal.OpBlock;
  req: DiskChannel.IORequestHandle;
  status: FloppyChannel.Status;
  Process.SetPriority[ProcessPriorities.diskInterruptPriority];
  DO
    -- main loop of request-processing process, which cannot be inside the monitor since it may
**take page faults
    req ← GetNextRequest[]; --monitored wait
    drive ← DiskChannelBackend.GetDrive[req.channel];
    labCountDone ← CheckLabels[req, drive];
    op ←
      [device: MakeHandle[drive],
        function:
        SELECT req.command FROM
          vvr, vrr => readSector,
          vvw, vww => writeSector,
          ENDCASE => ErrorHalt1[], address: LOOPHOLE[req.address],
        buffer:
        [Utilities.LongPointerFromPage[req.memoryPage],
          PilotFloppyFormat.wordsPerSector*labCountDone],
        incrementDataPtr: ~req.dontIncrement, count: labCountDone];
    [status, countDone] ← FloppyChannelInternal.DoRequest[@op];
    req.status ←
      IF ((labCountDone < req.count) AND (countDone = labCountDone)) THEN
      labelDoesNotMatch ELSE DecodeStatus[status];
    req.countDone ← countDone;
    DiskChannelBackend.NotifyIOComplete[req]
    ENDLOOP;
  END;
-- ENTRY PROCEDURES

RequestIO: ENTRY DiskChannelBackend.RequestIOProc =
  BEGIN
  -- Link request into IORequest list using unused link field
  req.address ← GetPageAddress[
    DiskChannelBackend.GetDrive[req.channel], req.diskPage];
  req.retryCount ← 0;
  req.next ← NIL;
  IF first ~= NIL THEN last.next ← req ELSE first ← req;
```



```
  last ← req;
  NOTIFY requestSubmitted;
  END;

GetNextRequest: ENTRY PROCEDURE RETURNS [req: DiskChannel.IORequestHandle] =
  -- Get the next request off the chain of unexamined IORequests
  --INLINE
  BEGIN
  WHILE first = NIL DO WAIT requestSubmitted ENDLOOP;
  req ← first;
  first ← first.next;
  END;
-- UNMONITORED PROCEDURES

CheckLabels: PROCEDURE [
  req: DiskChannel.IORequestHandle, drive: DiskChannelBackend.DriveHandle]
  RETURNS [run: CARDINAL] =
  -- Do page label processing for a request
  BEGIN
  i, j: PilotFloppyFormat.FloppyPageNumber;
  pLabels: PLabels;
  pLO: LONG POINTER TO LabelObject ← drive.driverStorage;
  -- The Following statement goes away when InitLabelImage and ChangeState return errors. C
**urrently a problem encountered initializing the label image is not known until an attempt is mad
**e to access labels.
  IF NOT pLO.hasLabels THEN RETURN[0];
  pLabels ← pLO.pLabels;
  i ← FloppyPage[req.diskPage];
  run ← Inline.LowHalf[req.count];
  -- reduce run to maximum # pages if too big?
  FOR j IN [0..run) DO
    SELECT req.command FROM
      vrr => req.label↑ ← pLabels[i + j];
      vww =>
        BEGIN pLabels[i + j] ← req.label↑; PilotDisk.NextLabel[req.label]; END;
      ENDCASE => -- vvr, vvw --
        BEGIN
        IF NOT PilotDisk.MatchLabels[req.label, @pLabels[i + j]] THEN RETURN[j];
        PilotDisk.NextLabel[req.label];
        END;
    ENDLOOP;
  IF req.command = vww THEN
    [labelsDone: run] ← UpdateLabelImage[drive, pLabels, i, run];
  END;

DecodeStatus: PROCEDURE [s: FloppyChannel.Status]
  RETURNS [status: DiskChannel.CompletionStatus] =
  BEGIN
  badMask: FloppyChannel.Status =
    [ --052400B--diskChanged: FALSE, tbd1: TRUE, twoSided: FALSE, tbd2: TRUE,
      error: FALSE, inProgress: TRUE, recalibrateError: FALSE,
      wrongSizeBuffer: FALSE, notReady: FALSE, writeProtect: FALSE,
      deletedData: FALSE, recordNotFound: FALSE, crcError: FALSE,
      track00: FALSE, hardwareError: FALSE, goodCompletion: FALSE];
  infoMask: FloppyChannel.Status =
    [ --057733B--diskChanged: FALSE, tbd1: TRUE, twoSided: FALSE, tbd2: TRUE,
      error: TRUE, inProgress: TRUE, recalibrateError: TRUE,
      wrongSizeBuffer: TRUE, notReady: TRUE, writeProtect: TRUE,
```

```
      deletedData: FALSE, recordNotFound: TRUE, crcError: TRUE, track00: FALSE,
      hardwareError: TRUE, goodCompletion: TRUE];
    IF LOOPHOLE[Inline.BITAND[s, badMask], CARDINAL] # 0 THEN ERROR;
    -- impossible statuses
    status ← LOOPHOLE[Inline.BITAND[s, infoMask]];
    -- mask off information statuses
    RETURN[
      SELECT TRUE FROM
        s.notReady => notReady,
        s.recordNotFound, s.wrongSizeBuffer => noSuchPage,
        s.crcError => checksumError,
        s.recalibrateError => seekFailed, --new error: recalibrateError?

        s.goodCompletion => goodCompletion
        ENDCASE => hardwareError];
    END;

  FloppyPage: PROCEDURE [dp: DiskChannel.DiskPageNumber]
    RETURNS [fp: PilotFloppyFormat.FloppyPageNumber] =
    -- Convert DiskChannel.DiskPageNumber to (short) FloppyPageNumber
    INLINE
    BEGIN
    -- check other half = 0
    RETURN[Inline.LowHalf[dp]];
    END;

  MakeHandle: PROCEDURE [dH: DiskChannelBackend.DriveHandle]
    RETURNS [h: DiskChannel.PVHandle] =
    -- Construct a PhysicalVolume Handle from a Drive Handle
    INLINE BEGIN RETURN[[LOOPHOLE[dH], dH.changeCount]] END;

  UpdateLabelImage: PROCEDURE [
    drive: DiskChannelBackend.DriveHandle, pLabels: PLabels,
    firstLabel: PilotFloppyFormat.FloppyPageNumber, count: CARDINAL]
    RETURNS [status: FloppyChannel.Status, labelsDone: CARDINAL] =
    BEGIN
    op: FloppyChannelInternal.OpBlock;
    firstPage, lastPage: PilotFloppyFormat.FloppyPageNumber;
    pageBase: Space.PageNumber = Space.PageFromLongPointer[pLabels];
    pageCount, pagesDone: CARDINAL;
    pageSize: CARDINAL = PilotFloppyFormat.wordsPerSector;
    firstPage ← (firstLabel*labelSize)/pageSize; --page containing first label
    lastPage ← (firstLabel + count - 1)*labelSize/pageSize;
    --page containing last label
    pageCount ← lastPage - firstPage + 1;
    op ←
      [device: MakeHandle[drive], function: writeSector,
        address: LOOPHOLE[GetDiskAddress[
        drive, PilotFloppyFormat.labelBasePage + firstPage]],
        buffer: [pLabels + firstPage*pageSize, pageSize*pageCount],
        incrementDataPtr: TRUE, count: pageCount];
    FOR i: Space.PageOffset IN [firstPage..lastPage] DO
      -- pin label pages involved
      SpecialSpace.MakeResident[Space.GetHandle[pageBase + i]];
      ENDLOOP;
    [status, pagesDone] ← FloppyChannelInternal.DoRequest[@op];
    FOR i: Space.PageOffset IN [firstPage..lastPage] DO
      -- unpin label pages
```



```
      SpecialSpace.MakeSwappable[Space.GetHandle[pageBase + i]];
      ENDLOOP;
    labelsDone ← IF status.goodCompletion THEN count ELSE 0;
    -- all or nothing right now
    -- to be supplied: labelsDone = # labels in pagesDone less padding before firstLabel and after
  **firstLabel + count-1
    -- refit op for second label area; repeat DoRequest
    -- what happens if error?

    END;
  -- INITIALIZATION PROCEDURES

  InitLabelImage: PROCEDURE [drive: DiskChannelBackend.DriveHandle] =
    -- Set up the labels image in virtual memory for the given drive
    BEGIN
    op: FloppyChannelInternal.OpBlock;
    pagesToHoldLabels: CARDINAL = Inline.LongDiv[
      drive.nPages*labelSize + Environment.wordsPerPage - 1,
      Environment.wordsPerPage];
    pLO: LONG POINTER TO LabelObject ← drive.driverStorage;
    sH: Space.Handle ← Space.Create[
      parent: Space.virtualMemory, size: pagesToHoldLabels];
    FOR base: Space.PageOffset IN [0..pagesToHoldLabels) DO
      -- Creating subspaces in order to be able to pin subsets of the label image
      [] ← Space.Create[parent: sH, size: 1, base: base];
      ENDLOOP;
    Space.Map[sH];
    SpecialSpace.MakeResident[sH];
    pLO.sH ← sH;
    pLO.pLabels ← Space.LongPointer[sH]; -- pointer to VM copy of labels
    op ←
      [device: MakeHandle[drive], function: readSector,
        address: LOOPHOLE[GetDiskAddress[drive, PilotFloppyFormat.labelBasePage]],
        buffer: [pLO.pLabels, pagesToHoldLabels*PilotFloppyFormat.wordsPerSector],
        incrementDataPtr: TRUE, count: pagesToHoldLabels];
    pLO.hasLabels ← FloppyChannelInternal.DoRequest[@op].status.goodCompletion;
    -- This method of handling the error in setting up the labels will change to return of a status (of
  **some kind) to caller ( = ChangeState procedure), which will then return a status of sorts. The bo
  **olean part of pLO will go away completely.
    SpecialSpace.MakeSwappable[sH];
    END;

  RegisterDriverProcs: PROCEDURE =
    -- Add Pilot-only elements of drive objects registered by FloppyChannel
    BEGIN
    device: DiskChannel.Drive ← DiskChannel.nullDrive;
    drive: DiskChannelBackend.DriveHandle;
    WHILE nDrives < maxDrives AND
      ((device ← DiskChannel.GetNextDrive[device]) ~= DiskChannel.nullDrive) DO
      drive ← LOOPHOLE[device];
      IF drive.driveID.type = DeviceTypes.sa800 THEN
        BEGIN
        drive.requestIO ← RequestIO;
        drive.getPageAddress ← GetPageAddress;
        drive.getPageNumber ← GetPageNumber;
        drive.changeState ← PilotChangeState;
        drive.driverStorage ← @labelBlock[nDrives]; -- label information block
        nDrives ← nDrives + 1;
```

```
        END;
      ENDLOOP;
    END;

  Start: PUBLIC PROCEDURE = -- exported to (StoreDriver)StartChain
    BEGIN RemainingDrivers.Start[]; END;
  -- INITIALIZATION

  RegisterDriverProcs[];
  IF nDrives > 0 THEN
    BEGIN
    Process.InitializeCondition[@requestSubmitted, Process.SecondsToTicks[60]];
    -- Make timeout long
    Process.Detach[FORK RequestProcess];
    -- Fork and start the request-handler process
    SpecialSpace.MakeCodeResident[SA800Impl]; -- Make us stay resident

    END;
  END.
LOG
Time: June 20, 1980 6:35 PM    By: Jose   Action: Create file
Time: September 15, 1980 2:55 PM          By: Jose   Action: Separate startup- and new volume-in
**itialization; pin code only if drive present
Time: October 10, 1980 3:23 PM            By: Jose   Action: Changes for runs of pages in head: d
**ivide label image into subspaces, pin when updating.
Time: October 21, 1980 2:15 PM            By: Jose   Action: Remove wrongSizeBuffer from impo
**ssible errors.
```

```
-- StartChainPlug.mesa (last edited by: McJones on: February 9, 1980  3:05 PM)

DIRECTORY
  DriverStartChain: FROM "DriverStartChain",
  HeadStartChain: FROM "HeadStartChain",
  StoreDriverStartChain: FROM "StoreDriverStartChain";

StartChainPlug: PROGRAM
  EXPORTS DriverStartChain, HeadStartChain, StoreDriverStartChain =
  BEGIN
  Start: PUBLIC PROCEDURE = BEGIN END;

  END.
```

LOG

Time: February 8, 1980 4:12 PM *By: Forrest* *Action: Create file*
Time: February 9, 1980 3:05 PM *By: McJones* *Action: Export to all three start ch*
***ains*

```
-- DialupImpl.mesa (last edited by Danielson: July 10, 1980 11:15 AM) --
DIRECTORY
  Dialup: FROM "Dialup" USING [Outcome, RetryCount],
  Process: FROM "Process" USING [MsecToTicks, SetTimeout],
  RS366Face: FROM "RS366Face" USING [
    GetDialerCount, GetStatus, SetStatus, GetStatusBits, SetStatusBits];

DialupImpl: MONITOR IMPORTS Process, RS366Face EXPORTS Dialup =
  BEGIN
  -- various definitions
  timer, oneTick: CONDITION;
  abortRequested: PACKED ARRAY [0..15] OF BOOLEAN ← ALL[FALSE];
  -- procedures (listed alphabetically)
  AbortCall: PUBLIC ENTRY PROCEDURE [dialerNumber: CARDINAL] =
    BEGIN abortRequested[dialerNumber] ← TRUE; END;

  Dial: PUBLIC ENTRY PROCEDURE [
    dialerNumber: CARDINAL, number: STRING, retries: Dialup.RetryCount]
    RETURNS [outcome: Dialup.Outcome] =
    BEGIN
    stringIndex: CARDINAL;
    setStatusBits: RS366Face.SetStatusBits;
    getStatusBits: RS366Face.GetStatusBits;
    abortRequested[dialerNumber] ← FALSE;
    -- Validate parameters --
    IF RS366Face.GetDialerCount[] <= dialerNumber THEN RETURN[dialerNotPresent];
    IF number.length > 31 THEN RETURN[formatError];
    -- Make certain that the dialer has power --
    IF ~RS366Face.GetStatus[dialerNumber].powerIndication THEN
      RETURN[dialerNotPresent];
    BEGIN
    THROUGH [0..retries] DO
      -- Dial the phone number retries-1 times
      setStatusBits ← ResetDialer[dialerNumber];
      IF (RS366Face.GetStatus[dialerNumber]).dataLineOccupied THEN
        RETURN[dataLineOccupied];
      setStatusBits.callRequest ← TRUE;
      RS366Face.SetStatus[dialerNumber, setStatusBits];
      BEGIN
      FOR stringIndex IN [0..number.length) DO
        THROUGH [0..200) DO
          -- 10 second timeout
          getStatusBits ← RS366Face.GetStatus[dialerNumber];
          IF getStatusBits.presentNextDigit THEN EXIT;
          IF getStatusBits.abandonCallAndRetry THEN GO TO retry;
          IF abortRequested[dialerNumber] THEN GO TO abortDialing;
          WAIT oneTick;
          REPEAT FINISHED => GOTO dialingTimeout;
          ENDLOOP;
        SELECT number[stringIndex] FROM
          IN ['0..'9] => setStatusBits.digit ← LOOPHOLE[number[stringIndex] - '0];
          '* => setStatusBits.digit ← 10;
          '# => setStatusBits.digit ← 11;
          '= => setStatusBits.digit ← 12; -- EON --
          '< => setStatusBits.digit ← 13; -- SEP..await dial tone --
          '> => -- timeout..pause before next digit --
            BEGIN WaitFor[6000]; LOOP; END;
          ENDCASE => GOTO formatError;
        RS366Face.SetStatus[dialerNumber, setStatusBits];
```



```
        setStatusBits.digitPresent ← TRUE;
        RS366Face.SetStatus[dialerNumber, setStatusBits];
        THROUGH [0..200) DO
          -- 10 second timeout
          getStatusBits ← RS366Face.GetStatus[dialerNumber];
          IF ~getStatusBits.presentNextDigit THEN EXIT;
          IF getStatusBits.abandonCallAndRetry THEN GO TO retry;
          IF abortRequested[dialerNumber] THEN GO TO abortDialing;
          WAIT oneTick;
          REPEAT FINISHED => GOTO dialingTimeout;
          ENDLOOP;
        setStatusBits.digitPresent ← FALSE;
        RS366Face.SetStatus[dialerNumber, setStatusBits];
        ENDLOOP; -- go on to next digit
      -- Wait until dialer has transferred control of communication line to the data set.
      THROUGH [0..2000) DO
        --100 second timeout
        getStatusBits ← RS366Face.GetStatus[dialerNumber];
        IF getStatusBits.callOriginationStatus THEN EXIT;
        IF getStatusBits.abandonCallAndRetry THEN GO TO retry;
        IF abortRequested[dialerNumber] THEN GOTO abortDialing;
        WAIT oneTick;
        REPEAT FINISHED => GOTO transferTimeout;
        ENDLOOP;
      -- Modem now in control...idle dialer --
      setStatusBits ← [FALSE, FALSE, 0];
      RS366Face.SetStatus[dialerNumber, setStatusBits];
      RETURN[success];
      EXITS retry => NULL;
      END;
      ENDLOOP;
    outcome ← failure;
    EXITS
      abortDialing => outcome ← aborted;
      dialingTimeout => outcome ← dialingTimeout;
      formatError => outcome ← formatError;
      transferTimeout => outcome ← transferTimeout;
    END;
    [] ← ResetDialer[dialerNumber];
    RETURN;
    END;

  ResetDialer: PRIVATE INTERNAL PROCEDURE [dialerNumber: CARDINAL]
    RETURNS [setStatusBits: RS366Face.SetStatusBits] =
    BEGIN -- Resets setStatusBits and hangs up phone --
    setStatusBits ← [callRequest: FALSE, digitPresent: FALSE, digit: 0];
    RS366Face.SetStatus[dialerNumber, setStatusBits];
    THROUGH [0..200) DO
      -- 10 second timeout for phone to be hung up --
      IF ~(RS366Face.GetStatus[dialerNumber]).dataLineOccupied THEN RETURN;
      WAIT oneTick;
      ENDLOOP;
    END;

  WaitFor: PRIVATE INTERNAL PROCEDURE [milliseconds: CARDINAL] =
    BEGIN
    Process.SetTimeout[@timer, Process.MsecToTicks[milliseconds]];
    WAIT timer;
```

```
    END;

  Process.SetTimeout[@oneTick, 1];
  END.
```

LOG
Time: October 18, 1978 8:47 AM By: Schwartz Action: Created file
Time: May 30, 1980 3:00 PM By: Schwartz Action: Remove all pre-Amargosa log entries
***, and change abortRequested to an array to allow support of multiple dialers.*
Time: July 10, 1980 10:55 AM By: Danielson Action: Modified to support use RS366Face i
***nterface.*

```
-- RS232CManagerImpl.mesa (last edited by: Forrest on: September 19, 1980 6:18 PM) --
DIRECTORY
  Heap USING [MakeNode],
  OISTransporter USING [Destroy, Initialize],
  Process USING [InitializeCondition],
  RS232C USING [
    AutoRecognitionOutcome, AutoRecognitionWait, ChannelAlreadyExists, Create,
    Delete, DeviceStatus, GetStatus, InvalidLineNumber, NoRS232CHardware, Suspend,
    UnimplementedFeature],
  RS232CFace USING [GetLineCount],
  RS232CInternal USING [ChannelStatusHandle],
  RS232CManager USING [
    CommParamHandle, CommParamObject, NetAccess, ReserveType, ChannelUseType,
    ReserveFailedReason];

RS232CManagerImpl: MONITOR
  IMPORTS Heap, OISTransporter, Process, RS232C, RS232CFace
  EXPORTS RS232C, RS232CManager =
  BEGIN
  -- types
  ChannelHandle: PUBLIC TYPE = RS232CInternal.ChannelStatusHandle;
  ChannelState: TYPE = RECORD [
    backgroundOIS: BOOLEAN,
    -- tells us whether to start OISTransporter when channel not in use
    undefinedUse: BOOLEAN, -- used for limbo state of reserved for auto-recog.
    reservedChHandle: ChannelHandle,
    lastPreemptMe: RS232CManager.ReserveType,
    channelUse: RS232CManager.ChannelUseType,
    currentLineParams: RS232CManager.CommParamObject,
    channelBeingReleased: CONDITION];
  -- writeable data
  channelStates: LONG DESCRIPTOR FOR ARRAY OF ChannelState;
  tempIndex: CARDINAL;
  -- The following default line params are used only until the first call to DescribeCommEquipment
**for the line.
  defaultLineParams: RS232CManager.CommParamObject ←
    [duplex: full, lineType: bitSynchronous, lineSpeed: bps1200,
      accessDetail: directConn[]];
  -- This is constant info. Assigned with ← rather than = to allow indirect reference.
  -- errors and signals
  ReserveFailed: PUBLIC ERROR [reason: RS232CManager.ReserveFailedReason] = CODE;
  NonrecoverableSoftwareFailure: PUBLIC ERROR = CODE;
  AwaitAutoRecognition: PUBLIC PROCEDURE [channel: ChannelHandle]
    RETURNS [correspondent: RS232C.AutoRecognitionOutcome] =
    -- wait for auto-recognition of caller
    BEGIN correspondent ← RS232C.AutoRecognitionWait[channel]; END;

  ConvertChannelHandleToLineNumber: PROCEDURE [channel: ChannelHandle]
    RETURNS [CARDINAL] =
    BEGIN
    index: CARDINAL;
    FOR index IN [0..LENGTH[channelStates]) DO
      IF channel = channelStates[index].reservedChHandle THEN RETURN[index];
      ENDLOOP;
    ERROR NonrecoverableSoftwareFailure;
    END;

  DefineIdleChannelUse: PUBLIC PROCEDURE [
```



```
    channel: ChannelHandle, doOisCommInBackground: BOOLEAN] =
    -- set flag to enable/disable background OISCP
    BEGIN
    channelStates[ConvertChannelHandleToLineNumber[channel]].backgroundOIS ←
      doOisCommInBackground;
    END;

  DescribeCommEquipment: PUBLIC PROCEDURE [
    portID: CARDINAL, commParamHandle: RS232CManager.CommParamHandle] =
    -- incorporate new set of comm line parameters
    BEGIN
    IF portID >= LENGTH[channelStates] THEN ERROR NonrecoverableSoftwareFailure;
    -- set channel parameters
    channelStates[portID].currentLineParams ← commParamHandle↑;
    END;

  GetCommEquipmentDescription: PUBLIC PROCEDURE [portID: CARDINAL]
    RETURNS [RS232CManager.CommParamObject] =
    -- return current set of comm line parameters
    BEGIN
    IF portID >= LENGTH[channelStates] THEN ERROR NonrecoverableSoftwareFailure;
    RETURN[channelStates[portID].currentLineParams];
    END;

  RedefineChannelUse: PUBLIC ENTRY PROCEDURE [
    channel: ChannelHandle, useType: RS232CManager.ChannelUseType,
    preemptOthers, preemptMe: RS232CManager.ReserveType] =
    -- allow client to redefine use after waiting for auto-recognition
    BEGIN
    ENABLE UNWIND => NULL; -- release monitor lock
    -- locals
    currentLine: CARDINAL ← ConvertChannelHandleToLineNumber[channel];
    -- allow redefinition only after auto-recog
    IF channelStates[currentLine].undefinedUse THEN
      BEGIN
      channelStates[currentLine].lastPreemptMe ← preemptMe;
      -- we rely on client to set the line type correctly !!
      IF (channelStates[currentLine].channelUse ← useType) = oisCommunication THEN
        OISTransporter.Initialize[
          channel, channelStates[currentLine].currentLineParams];
      channelStates[currentLine].undefinedUse ← FALSE;
      END;
    END;

  ReleaseChannel: PUBLIC ENTRY PROCEDURE [channel: ChannelHandle] =
    -- release the channel, start OIS listener if necessary
    BEGIN
    ENABLE UNWIND => NULL; -- release monitor lock
    -- locals
    currentLine: CARDINAL ← ConvertChannelHandleToLineNumber[channel];
    -- If the channel is currently being used for OIS Communication, turn off OIS Transporter
    IF channelStates[currentLine].channelUse = oisCommunication THEN
      OISTransporter.Destroy[channel];
    -- Delete the channel
    RS232C.Delete[channel];
    channelStates[currentLine].reservedChHandle ← NIL;
    NOTIFY channelStates[currentLine].channelBeingReleased;
    -- inform process waiting in ReserveChannel
    -- now go back to default background activity (OIS)
```

```
  IF channelStates[currentLine].backgroundOIS THEN
    BEGIN
    -- Create an OIS-type channel
    channelStates[currentLine].reservedChHandle ← RS232C.Create[
      0, channelStates[currentLine].currentLineParams.lineType !
      RS232C.ChannelAlreadyExists => ERROR NonrecoverableSoftwareFailure];
    -- Start OIS transporter
    channelStates[currentLine].channelUse ← oisCommunication;
    channelStates[currentLine].lastPreemptMe ← preemptInactive;
    OISTransporter.Initialize[
      channelStates[currentLine].reservedChHandle, channelStates[
      currentLine].currentLineParams];
    END;
  END;

ReserveChannel: PUBLIC ENTRY PROCEDURE [
  portID: CARDINAL, useType: RS232CManager.ChannelUseType,
  preemptOthers, preemptMe: RS232CManager.ReserveType,
  commParamHandle: RS232CManager.CommParamHandle] RETURNS [ChannelHandle] =
  -- reserve a channel
  --          Old Channel Owner's preemptMe
  --                              Never    If Inactive    Always
  -- New      Never               Don't      Don't        Don't
  -- preempt- If Inactive         Don't      Preempt      Preempt
  -- Others   Always              Don't      Preempt      Preempt
  BEGIN OPEN RS232CManager;
  -- locals
  chStatus: RS232C.DeviceStatus;
  preempt: BOOLEAN ← TRUE;
  BEGIN
  ENABLE UNWIND => NULL; -- release monitor lock
  IF portID >= LENGTH[channelStates] THEN
    ERROR ReserveFailed[reason: noRS232CHardware];
  -- Check for consistency of specified params with current ones
  IF
    ~(commParamHandle↑.netAccess = channelStates[
      portID].currentLineParams.netAccess) AND
    (commParamHandle↑.duplex = channelStates[portID].currentLineParams.duplex)
    AND
    (commParamHandle↑.lineType = channelStates[
      portID].currentLineParams.lineType) AND
    (commParamHandle↑.lineSpeed = channelStates[
      portID].currentLineParams.lineSpeed) -- should check variants perhaps? --
    THEN ERROR ReserveFailed[reason: inconsistentParams];
  -- Create channel if not already created
  IF channelStates[portID].reservedChHandle # NIL THEN
    BEGIN
    preempt ← FALSE;
    SELECT preemptOthers FROM
      preemptNever => NULL; -- fail

      preemptInactive =>
        BEGIN
        SELECT channelStates[portID].lastPreemptMe FROM
          preemptInactive =>
            BEGIN
            -- if inactive, delete old channel
            chStatus ← RS232C.GetStatus[channelStates[portID].reservedChHandle];
            IF ~chStatus.dataSetReady THEN preempt ← TRUE;
```



```
            END;
          preemptAlways => preempt ← TRUE;
          ENDCASE => NULL; -- fail

        END;
      preemptAlways =>
        IF channelStates[portID].lastPreemptMe # preemptNever THEN
          preempt ← TRUE;
      ENDCASE;
    IF ~preempt THEN ERROR ReserveFailed[reason: inUse]
    ELSE
      BEGIN
      RS232C.Suspend[channelStates[portID].reservedChHandle, all];
      -- Give the channel user a STRONG HINT to release channel at once.
      UNTIL channelStates[portID].reservedChHandle = NIL DO
        WAIT channelStates[portID].channelBeingReleased; ENDLOOP;
      END;
    END;
  -- set last preemptMe
  channelStates[portID].lastPreemptMe ← preemptMe;
  -- Create New Channel
  channelStates[portID].reservedChHandle ← RS232C.Create[
    portID, commParamHandle.lineType !
    RS232C.ChannelAlreadyExists =>
      BEGIN -- this should not happen
      channelStates[portID].reservedChHandle ← NIL;
      ERROR ReserveFailed[reason: inUse];
      END;
    RS232C.NoRS232CHardware, RS232C.InvalidLineNumber =>
      BEGIN
      channelStates[portID].reservedChHandle ← NIL;
      ERROR ReserveFailed[reason: noRS232CHardware];
      END;
    RS232C.UnimplementedFeature =>
      BEGIN -- only known one is auto-reg
      channelStates[portID].reservedChHandle ← NIL;
      ERROR ReserveFailed[reason: unimplementedFeature];
      END];
  channelStates[portID].undefinedUse ←
    (commParamHandle.lineType = autoRecognition);
  -- set channel type flag and start Transporter if necessary
  IF (channelStates[portID].channelUse ← useType) = oisCommunication THEN
    BEGIN -- ignore useType for auto-recognition
    IF ~channelStates[portID].undefinedUse THEN
      OISTransporter.Initialize[
        channelStates[portID].reservedChHandle, commParamHandle↑];
    END;
  END; -- disable UNWIND catch
  RETURN[channelStates[portID].reservedChHandle];
  END;
-- MAIN PROGRAM --
-- Allocate a channelState record for each line, and initialize them all to defaults.

channelStates ← DESCRIPTOR[
  Heap.MakeNode[, RS232CFace.GetLineCount[]*SIZE[ChannelState]],
    RS232CFace.GetLineCount[]];
FOR tempIndex IN [0..LENGTH[channelStates]) DO
  OPEN channelStates[tempIndex];
```

```
    -- channelStates[tempIndex] ← [backgroundOIS: FALSE, undefinedUse: FALSE, reservedChH
**andle: NIL, lastPreemptMe: preemptAlways, channelUse: foreignDeviceAccess, currentLinePar
**ams: defaultLineParams, channelBeingReleased: ];
    backgroundOIS ← FALSE;
    undefinedUse ← FALSE;
    reservedChHandle ← NIL;
    lastPreemptMe ← preemptAlways;
    channelUse ← foreignDeviceAccess;
    currentLineParams ← defaultLineParams;
    Process.InitializeCondition[@channelBeingReleased, 800]; -- 20 second timeout

    ENDLOOP;
  END.
LOG
Time: June 24, 1980 3:20 PM    By: Schwartz        Action: Removed pre-Amargosa log entries.
Time: June 24, 1980 3:20 PM    By: Schwartz        Action: Adapted to RS232C changes relating
**to Abort/Suspend/Resume, and channelHandle validity checks.
Time: June 26, 1980 2:30 PM    By: Schwartz        Action: Add channelHandle parameter to De
**fineIdleChannelUse.
Time: July 9, 1980 12:11 PM    By: Schwartz        Action: Handle multiple RS232C lines and ha
**ndle preemption via Suspend rather than Delete.
Time: August 4, 1980 4:15 PM   By: Schwartz        Action: Use Heap, rather than RS232CHeap
**for storage allocation.
Time: August 5, 1980 5:30 PM   By: Schwartz        Action: Destroy the OIS Transporter when th
**e channel is released.
Time: September 19, 1980 6:10 PM    By: Forrest        Action: Change to use new compi
**ler prohibitng updating of records containing MONITORS and CONDITIONS.
```

```
-- File: RS232CDriverA.mesa
-- LastEdited: September 19, 1980 5:38 PM  By: Forrest
-- This is the Monitor for RS232C Channel creation/deletion (i.e. for operations which involve data
**not contained in the RS232C ChannelStatus record.

DIRECTORY
  Heap USING [FreeNode, MakeNode],
  Process USING [InitializeMonitor],
  RS232C USING [LineType],
  RS232CFace USING [GetLineCount, Off, On, ResetLine],
  RS232CInternal USING [
    asynchronousDefaults, autoRecognitionDefaults, bitSynchronousDefaults,
    byteSynchronousDefaults, ChannelStatus, ChannelStatusHandle, DeleteChannel];

RS232CDriverA: MONITOR
  IMPORTS Heap, Process, RS232CFace, RS232CInternal EXPORTS RS232C, RS232CInternal
  =
  BEGIN
  ChannelHandle: PUBLIC TYPE = RS232CInternal.ChannelStatusHandle;
  lineCount: CARDINAL;
  channelIsCreated: PUBLIC PACKED ARRAY [0..15] OF BOOLEAN ← ALL[FALSE];
  ChannelAlreadyExists: PUBLIC ERROR = CODE;
  InvalidLineNumber: PUBLIC ERROR = CODE;
  NoRS232CHardware: PUBLIC ERROR = CODE;
  UnimplementedFeature: PUBLIC ERROR = CODE;
  Create: PUBLIC ENTRY PROCEDURE [lineNumber: CARDINAL, lineType: RS232C.LineType]
    RETURNS [channel: ChannelHandle] =
    BEGIN
    ENABLE UNWIND => IF channel # NIL THEN Heap.FreeNode[, channel];
    -- Required in order to release monitor lock, and to return block to heap
    channel ← NIL;
    IF lineCount = 0 THEN ERROR NoRS232CHardware;
    IF lineCount - 1 < lineNumber THEN ERROR InvalidLineNumber;
    IF channelIsCreated[lineNumber] THEN ERROR ChannelAlreadyExists;
    -- Allocate and initialize a ChannelStatus record
    channel ← Heap.MakeNode[, SIZE[RS232CInternal.ChannelStatus]];
    -- channel↑ ← [ + + monitor lock + + , , FALSE, FALSE, FALSE, lineNumber, 0];
    Process.InitializeMonitor[@channel.LOCK];
    -- Initialize the MONITOR in the channel status record
    channel.parameterRecord ←
      SELECT lineType FROM
        bitSynchronous => RS232CInternal.bitSynchronousDefaults,
        byteSynchronous => RS232CInternal.byteSynchronousDefaults,
        asynchronous => RS232CInternal.asynchronousDefaults,
        autoRecognition => RS232CInternal.autoRecognitionDefaults,
        ENDCASE => ERROR; -- This should be logged as a system error.
    channel.inputSuspended ← channel.outputSuspended ← channel.otherSuspended ←
      FALSE;
    channel.lineNumber ← lineNumber;
    channel.statusWaitCount ← 0;
    RS232CFace.On[lineNumber];
    IF RS232CFace.ResetLine[lineNumber, @channel.parameterRecord] # success THEN
      BEGIN
      RS232CFace.Off[lineNumber];
      ERROR UnimplementedFeature; -- (e.g. autoRecognition)

      END;
    channelIsCreated[lineNumber] ← TRUE;
```



```
    RETURN[channel];
    END;

  Delete: PUBLIC ENTRY PROCEDURE [channel: ChannelHandle] =
    BEGIN
    ENABLE UNWIND => IF channel # NIL THEN Heap.FreeNode[, channel];
    -- Required in order to release monitor lock, and to return block to heap
    -- Client is responsible for avoiding the passing of an OLD channel handle.
    RS232CInternal.DeleteChannel[channel];
    -- modify/release channel status record under object monitor
    channelIsCreated[channel.lineNumber] ← FALSE;
    RS232CFace.Off[channel.lineNumber];
    -- Release block to heap (cannot be done in DeleteChannel, since the MONITOR LOCK, which
**is in channel record, must still exist to be released at exit from DeleteChannel)
    Heap.FreeNode[, channel];
    END;
  -- MAIN PROGRAM --

  lineCount ← RS232CFace.GetLineCount[];
  IF lineCount > 16 THEN ERROR;
  -- Some tables are hard-coded to maximum size = 16. If there are ever more than 16 RS232C li
**nes on a single machine, a few changes will be necessary.

  END. -- RS232CDriverA
LOG
Time: May 23, 1980 12:01 PM  By: Victor Schwartz  Action: Pre-Amargosa log entries deleted.
Time: May 23, 1980 12:01 PM  By: Victor Schwartz  Action: Change name to RS232CDriverA (fro
**m RS232CFrontEnd) and start of modifications to handle multiple RS232C Channels.
Time: May 30, 1980 2:49 PM  By: Victor Schwartz  Action: Call RS232CFace.On and .Off on a li
**ne-by-line basis.
Time: July 1, 1980 2:48 PM  By: Victor Schwartz  Action: Change semantics associated with S
**uspend/Restart and handling of invalid channelHandles. Turn RS232CFace OFF if create fails.
Time: July 15, 1980 7:53 AM  By: Victor Schwartz  Action: Remove references to statusChange
**, a field which has been removed from the ChannelStatus record.
Time: August 4, 1980 2:36 PM  By: Victor Schwartz  Action: Use Heap, rather than RS232CHeap
**for storage allocation.
Time: September 19, 1980 5:35 PM  By: Forrest  Action: code around new compile
**r restriction concerning Monitored Records.
```

```
-- File: RS232CDriverB.mesa
-- LastEdited: August 5, 1980  11:22 AM  By: VSS
-- This is the Object Monitor for RS232C ChannelStatus records

DIRECTORY
  Process USING [SetTimeout],
  RS232C USING [
    AutoRecognitionOutcome, CompletionHandle, DeviceStatus, LineType,
    OperationClass, Parameter, PhysicalRecordHandle, TransferStatus,
    UnimplementedFeature],
  RS232CFace USING [
    AutoRecognitionWait, AbortInput, AbortOutput, AbortStatus, DeviceStatus, Get,
    GetStatus, Put, ResetLine, SendBreak, SetParameters, StatusWait,
    TransferStatus, TransferWait, TransmitNow],
  RS232CInternal USING [
    asynchronousDefaults, autoRecognitionDefaults, bitSynchronousDefaults,
    byteSynchronousDefaults, ChannelStatusHandle];

RS232CDriverB: MONITOR LOCKS channel USING channel: ChannelHandle
  IMPORTS Process, RS232C, RS232CFace EXPORTS RS232C, RS232CInternal =
  BEGIN
  ChannelHandle: PUBLIC TYPE = RS232CInternal.ChannelStatusHandle;
  ChannelSuspended: PUBLIC ERROR = CODE;
  InvalidParameter: PUBLIC ERROR = CODE;
  SendBreakIllegal: PUBLIC ERROR = CODE;
  aMoment: CONDITION;
  ConvertFaceToChannelTransferStatus: ARRAY RS232CFace.TransferStatus OF
    RS232C.TransferStatus =
    [success, dataLost, deviceError, frameTimeout, checksumError, parityError,
     asynchFramingError, invalidChar, invalidFrame, aborted, disaster];
  AutoRecognitionWait: PUBLIC PROCEDURE [channel: ChannelHandle]
    RETURNS [outcome: RS232C.AutoRecognitionOutcome] =
    BEGIN
    IF channel.otherSuspended THEN ERROR ChannelSuspended;
    outcome ← RS232CFace.AutoRecognitionWait[channel.lineNumber];
    END;

  DecrementStatusWaitCount: PRIVATE ENTRY PROCEDURE [channel: ChannelHandle] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    channel.statusWaitCount ← channel.statusWaitCount - 1;
    END;

  DeleteChannel: PUBLIC ENTRY PROCEDURE [channel: ChannelHandle] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    channel.inputSuspended ← channel.outputSuspended ← channel.otherSuspended ←
      TRUE;
    RS232CFace.AbortOutput[channel.lineNumber];
    RS232CFace.AbortInput[channel.lineNumber];
    RS232CFace.AbortStatus[channel.lineNumber];
    --This will complete calls to RS232CFace.StatusWait, and consequently, calls to RS232C.Statu
**sWait as well.
    channel.parameterRecord.dataTerminalReady ← FALSE;
    IF RS232CFace.SetParameters[channel.lineNumber, @channel.parameterRecord] #
      success THEN ERROR; -- This should be logged as a system error;
    UNTIL channel.statusWaitCount = 0 DO WAIT aMoment; ENDLOOP;
    -- This guaranteees that Delete has completed.
```



```
    END;

  Get: PUBLIC PROCEDURE [channel: ChannelHandle, rec: RS232C.PhysicalRecordHandle]
    RETURNS [RS232C.CompletionHandle] =
    BEGIN
    IF channel.inputSuspended THEN ERROR ChannelSuspended;
    RETURN[RS232CFace.Get[channel.lineNumber, rec]];
    END;

  GetStatus: PUBLIC PROCEDURE [channel: ChannelHandle]
    RETURNS [stat: RS232C.DeviceStatus] =
    BEGIN
    faceStat: RS232CFace.DeviceStatus;
    IF channel.otherSuspended THEN ERROR ChannelSuspended;
    faceStat ← RS232CFace.GetStatus[channel.lineNumber];
    RETURN[
      [FALSE, faceStat.dataLost, faceStat.breakDetected, faceStat.clearToSend,
       faceStat.dataSetReady, faceStat.carrierDetect, faceStat.ringHeard,
       faceStat.ringIndicator]];
    END;

  IncrementStatusWaitCount: PRIVATE ENTRY PROCEDURE [channel: ChannelHandle] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    channel.statusWaitCount ← channel.statusWaitCount + 1;
    END;

  Put: PUBLIC PROCEDURE [channel: ChannelHandle, rec: RS232C.PhysicalRecordHandle]
    RETURNS [RS232C.CompletionHandle] =
    BEGIN
    IF channel.outputSuspended THEN ERROR ChannelSuspended;
    RETURN[RS232CFace.Put[channel.lineNumber, rec]]
    END;

  Restart: PUBLIC ENTRY PROCEDURE [
    channel: ChannelHandle, class: RS232C.OperationClass] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    SELECT class FROM
      input => channel.inputSuspended ← FALSE;
      output => channel.outputSuspended ← FALSE;
      other => channel.otherSuspended ← FALSE;
      all =>
        channel.inputSuspended ← channel.inputSuspended ← channel.otherSuspended ←
          FALSE;
      ENDCASE => ERROR; -- This should be logged as a system error.

    END;

  SendBreak: PUBLIC PROCEDURE [channel: ChannelHandle] =
    BEGIN
    IF channel.outputSuspended THEN ERROR ChannelSuspended;
    IF channel.parameterRecord.lineType = byteSynchronous THEN
      ERROR SendBreakIllegal;
    RS232CFace.SendBreak[channel.lineNumber];
    END;

  SetLineType: PUBLIC ENTRY PROCEDURE [
    channel: ChannelHandle, lineType: RS232C.LineType] =
```

```
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    holdDataTerminalReady: BOOLEAN;
    IF channel.otherSuspended THEN ERROR ChannelSuspended;
    holdDataTerminalReady ← channel.parameterRecord.dataTerminalReady;
    channel.parameterRecord ←
      SELECT lineType FROM
        bitSynchronous => RS232CInternal.bitSynchronousDefaults,
        byteSynchronous => RS232CInternal.byteSynchronousDefaults,
        asynchronous => RS232CInternal.asynchronousDefaults,
        autoRecognition => RS232CInternal.autoRecognitionDefaults,
        ENDCASE => ERROR; -- This should be logged as a system error.
    channel.parameterRecord.dataTerminalReady ← holdDataTerminalReady;
    -- This will prevent disconnect, if DTR had been set prior to this call
    IF RS232CFace.ResetLine[channel.lineNumber, @channel.parameterRecord] #
      success THEN ERROR RS232C.UnimplementedFeature;
    END;

  SetParameter: PUBLIC ENTRY PROCEDURE [
    channel: ChannelHandle, parameter: RS232C.Parameter] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    IF channel.otherSuspended THEN ERROR ChannelSuspended;
    WITH parameter SELECT FROM
      charLength => channel.parameterRecord.charLength ← charLength;
      correspondent => channel.parameterRecord.correspondent ← correspondent;
      dataTerminalReady =>
        channel.parameterRecord.dataTerminalReady ← dataTerminalReady;
      frameTimeout => channel.parameterRecord.frameTimeout ← frameTimeout;
      latchBitClear =>
        BEGIN
        IF latchBitClearMask.dataLost THEN
          channel.parameterRecord.resetDataLost ← TRUE;
        IF latchBitClearMask.breakDetected THEN
          channel.parameterRecord.resetBreakDetected ← TRUE;
        IF latchBitClearMask.ringHeard THEN
          channel.parameterRecord.resetRingHeard ← TRUE;
        END;
      lineSpeed => channel.parameterRecord.lineSpeed ← lineSpeed;
      parity => channel.parameterRecord.parity ← parity;
      requestToSend => channel.parameterRecord.requestToSend ← requestToSend;
      stopBits => channel.parameterRecord.stopBits ← stopBits;
      syncChar => channel.parameterRecord.syncChar ← syncChar;
      syncCount => channel.parameterRecord.syncCount ← syncCount;
      ENDCASE => ERROR InvalidParameter;
    IF RS232CFace.SetParameters[channel.lineNumber, @channel.parameterRecord] #
      success THEN ERROR RS232C.UnimplementedFeature;
    channel.parameterRecord.resetDataLost ←
      channel.parameterRecord.resetBreakDetected ←
      channel.parameterRecord.resetRingHeard ← FALSE;
    -- for next call to SetParameters, make certain that latched bits don't get cleared again

    END;

  StatusWait: PUBLIC PROCEDURE [channel: ChannelHandle, stat: RS232C.DeviceStatus]
    RETURNS [newstat: RS232C.DeviceStatus] =
    BEGIN
    faceStat: RS232CFace.DeviceStatus;
```



```
    -- Note that the deletion of the channel subsequent to the call to StatusWait will cause StatusW
  **ait to return normally.
    IF channel.otherSuspended THEN ERROR ChannelSuspended;
    stat.statusAborted ← FALSE;
    IncrementStatusWaitCount[channel];
    DO
      --until newstat # stat
      faceStat ← RS232CFace.StatusWait[
        channel.lineNumber,
        [stat.dataLost, stat.breakDetected, stat.clearToSend, stat.dataSetReady,
          stat.carrierDetect, stat.ringHeard, stat.ringIndicator]];
      newstat ←
        [channel.otherSuspended, faceStat.dataLost, faceStat.breakDetected,
          faceStat.clearToSend, faceStat.dataSetReady, faceStat.carrierDetect,
          faceStat.ringHeard, faceStat.ringIndicator];
      IF newstat # stat THEN EXIT;
      ENDLOOP;
    DecrementStatusWaitCount[channel];
    RETURN;
    END;

  Suspend: PUBLIC ENTRY PROCEDURE [
    channel: ChannelHandle, class: RS232C.OperationClass] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    SELECT class FROM
      input =>
        BEGIN
        channel.inputSuspended ← TRUE;
        RS232CFace.AbortInput[channel.lineNumber];
        END;
      output =>
        BEGIN
        channel.outputSuspended ← TRUE;
        RS232CFace.AbortOutput[channel.lineNumber];
        END;
      other =>
        BEGIN
        channel.otherSuspended ← TRUE;
        RS232CFace.AbortStatus[channel.lineNumber];
        END;
      all =>
        BEGIN
        channel.inputSuspended ← channel.outputSuspended ← channel.otherSuspended
          ← TRUE;
        RS232CFace.AbortInput[channel.lineNumber];
        RS232CFace.AbortOutput[channel.lineNumber];
        RS232CFace.AbortStatus[channel.lineNumber];
        END;
      ENDCASE => ERROR; -- This should be logged as a system error.

    END;

  TransferWait: PUBLIC PROCEDURE [
    channel: ChannelHandle, event: RS232C.CompletionHandle]
    RETURNS [byteCount: CARDINAL, status: RS232C.TransferStatus] =
    BEGIN
    faceStatus: RS232CFace.TransferStatus;
    [byteCount, faceStatus] ← RS232CFace.TransferWait[channel.lineNumber, event];
```

```
    status ← ConvertFaceToChannelTransferStatus[faceStatus];
    RETURN;
    END;

  TransmitNow: PUBLIC PROCEDURE [
    channel: ChannelHandle, event: RS232C.CompletionHandle]
    RETURNS [byteCount: CARDINAL, status: RS232C.TransferStatus] =
    BEGIN
    faceStatus: RS232CFace.TransferStatus;
    [byteCount, faceStatus] ← RS232CFace.TransmitNow[channel.lineNumber, event];
    status ← ConvertFaceToChannelTransferStatus[faceStatus];
    RETURN;
    END;
  -- MAIN PROGRAM --

  Process.SetTimeout[@aMoment, 1];
  END. -- RS232CDriverB
LOG
Time: May 23, 1980  12:01 PM   By: Victor Schwartz   Action: Pre-Amargosa log entries deleted.
Time: May 23, 1980  12:01 PM   By: Victor Schwartz   Action: Start of modifications to handle multi
**ple RS232C Channels.
Time: June 20, 1980  2:45 PM   By: Victor Schwartz   Action: Change semantics of Suspend/Rest
**art.  Remove Abort.
Time: August 4, 1980  2:53 PM   By: Victor Schwartz   Action: Use Heap, rather than RS232CHeap
**to allocate storage.
```

```
DIRECTORY
  SpecialTransaction;

SpecialTransactionStub: PROGRAM EXPORTS SpecialTransaction =
  BEGIN
  ClientStart: PUBLIC PROCEDURE = {};

  END.
```

```
-- EthernetHeadD0.mesa (last edited by: HGM on: September 14, 1980 6:33 AM)

DIRECTORY
  Inline USING [LowHalf],
  HeadStartChain USING [Start],
  DeviceCleanup USING [Item, Await],
  D0InputOutput USING [
    CSB, IOPage, ethernetIn, ethernetOut, ControllerNumber, GetNextController,
    nullControllerNumber],
  Mopcodes USING [zMISC, zSTARTIO],
  SpecialSystem USING [ProcessorID],
  EthernetFace;

EthernetHeadD0: PROGRAM
  IMPORTS
    D0InputOutput, DeviceCleanup, RemainingStartChain: HeadStartChain, Inline
  EXPORTS EthernetFace, HeadStartChain =
  BEGIN OPEN EthernetFace;

  -- These are the data structures that the microcode knows about.  If some other module (for exa
**mple, a diagnostic) ever needs this info, it should probably get split out into a separate module.
**For now, it is lumped in here to avoid cluttering up the world with yet another file.

  OCSB: TYPE = LONG POINTER TO OutputControllerStatusBlock;
  OutputControllerStatusBlock: TYPE = MACHINE DEPENDENT RECORD [
    next: ShortIOCB,
    unused1: WORD,
    unused2: WORD,
    unused3: WORD,
    interruptBit: WORD, -- words after here are unused by microcode
    last: IOCB]; -- last IOCB on output queue, valid if next# noIOCB

  ICSB: TYPE = LONG POINTER TO InputControllerStatusBlock;
  InputControllerStatusBlock: TYPE = MACHINE DEPENDENT RECORD [
    next: ShortIOCB,
    host: SpecialSystem.ProcessorID,
    interruptBit: WORD,
    missed: WORD, -- for debugging only
    spare1: WORD,
    spare2: WORD,
    buffer: ARRAY [0..4) OF CARDINAL, -- words after here are unused by microcode
    last: IOCB]; -- last IOCB on input queue, valid if next# noIOCB

  IOCB: TYPE = LONG POINTER TO IOControlBlock;
  -- Beware that you don't automatically lengthen one of these.  If you do you will end up with a poi
**nter into your MDS rather then the first 64K where the IOCBs live.  That won't work unless your
**MDS is also the first 64K.
  ShortIOCB: TYPE = POINTER TO IOControlBlock;
  IOControlBlock: TYPE = MACHINE DEPENDENT RECORD [
    next: ShortIOCB,
    mask: WORD,
    spare: WORD,
    completion: WORD,
    used: CARDINAL, -- input only
    length: CARDINAL,
    buffer: LONG POINTER]; -- NB: Must be QuadWord Aligned

  StartIO: PROCEDURE [SioParameter] = MACHINE CODE BEGIN Mopcodes.zSTARTIO END;
```



```
  SioParameter: TYPE = RECORD [WORD];
  firstFixupOutput: SioParameter = [20B];
  firstFixupInput: SioParameter = [40B];
  firstReset: SioParameter = [60B];
  secondFixupOutput: SioParameter = [100B];
  secondFixupInput: SioParameter = [200B];
  secondReset: SioParameter = [300B];

  -- Input from reg 0 is device id

  Output: PROCEDURE [Command, Register] = MACHINE CODE
    BEGIN Mopcodes.zMISC, 6; END;

  Input: PROCEDURE [Register] RETURNS [WORD] = MACHINE CODE
    BEGIN Mopcodes.zMISC, 5; END;

  Command: TYPE = RECORD [WORD];
  enableInput: Command = [220B];
  enableOutput: Command = [103B];

  Register: TYPE = MACHINE DEPENDENT RECORD [
    zero: [0..377B] ← 0,
    controller: D0InputOutput.ControllerNumber,
    register: [0..17B]];

  -- completion bits
  processed: WORD = 040000B;
  error: WORD = 020000B;
  hardwareError: WORD = 010000B;
  fragment, tooLong: WORD = 004000B;
  loadOverflow: WORD = 002000B;
  nothingYet: WORD = 0;
  -- 001000 and 000400 are unused so far

  -- Hardware error bits:
  -- 001: Memory Data Fault (OFault)
  -- 002: Collision
  -- 004: Output Underrrun
  -- 010: Bad Parity (between mem and shifter)
  -- 020: CRC
  -- 040: Jam
  -- 100: Input Overrun
  -- 200: Bad Alignment

  noIOCB: ShortIOCB = LOOPHOLE[0];

  Device: TYPE = RECORD [board: Board, in, out: D0InputOutput.ControllerNumber];

  Board: TYPE = [0..2);

  ICSBFronDevice: PROCEDURE [device: Device] RETURNS [ICSB] = INLINE
    BEGIN RETURN[LOOPHOLE[@D0InputOutput.IOPage[device.in]]]; END;

  OCSBFronDevice: PROCEDURE [device: Device] RETURNS [OCSB] = INLINE
    BEGIN RETURN[LOOPHOLE[@D0InputOutput.IOPage[device.out]]]; END;

  ControlRegister: PROCEDURE [c: D0InputOutput.ControllerNumber]
    RETURNS [Register] = INLINE
```

```
  BEGIN
  RETURN[[0, c, 0]]; -- Register 0 is the control register

  END;

Shorten: PROCEDURE [iocb: IOCB] RETURNS [ShortIOCB] = INLINE
  BEGIN
  -- Maybe we should check to be sure that the high half is zero
  RETURN[Inline.LowHalf[iocb]];
  END;


-- EXPORTed TYPEs

DeviceHandle: PUBLIC TYPE = Device;
ControlBlockRecord: PUBLIC TYPE = IOControlBlock;

-- EXPORTed variables

nullDeviceHandle: PUBLIC DeviceHandle ← LOOPHOLE[123456B];
globalStateSize: PUBLIC CARDINAL ← 0;
controlBlockSize: PUBLIC CARDINAL ← SIZE[IOControlBlock];
hearSelf: PUBLIC BOOLEAN ← TRUE;


-- Non EXPORTed things.  Note that all the state information lives in the CSBs.

fixupInputBits: ARRAY Board OF SioParameter =
  [firstFixupInput, secondFixupInput];
fixupOutputBits: ARRAY Board OF SioParameter =
  [firstFixupOutput, secondFixupOutput];
resetBits: ARRAY Board OF SioParameter = [firstReset, secondReset];

QueueOutput: PUBLIC PROCEDURE [
  device: Device, buffer: LONG POINTER, length: CARDINAL, cb: IOCB] =
  BEGIN
  out: OCSB = OCSBFronDevice[device];
  cb↑ ←
    [next: nolOCB, mask: 0, spare: 0, completion: 0, used: 0, length: length,
      buffer: buffer];
  IF out.next = nolOCB THEN
    BEGIN -- new iocb, hardware idle
    out.next ← Shorten[cb];
    Output[enableOutput, ControlRegister[device.out]]; -- poke hardware

    END
  ELSE
    BEGIN -- output active, add to end of chain
    out.last.next ← Shorten[cb];
    IF out.next = nolOCB AND cb.completion = 0 THEN
      BEGIN -- oops, hardware went idle
      out.next ← Shorten[cb];
      Output[enableOutput, ControlRegister[device.out]]; -- poke hardware

      END;
    END;
  out.last ← cb;
  END;
```



```
QueueInput: PUBLIC PROCEDURE [
  device: Device, buffer: LONG POINTER, length: CARDINAL, cb: IOCB] =
  BEGIN
  in: ICSB = ICSBFronDevice[device];
  cb↑ ←
    [next: nolOCB, mask: 0, spare: 0, completion: 0, used: 0, length: length,
      buffer: buffer];
  IF in.next # nolOCB THEN in.last.next ← Shorten[cb];
  IF in.next = nolOCB AND cb.completion = 0 THEN
    BEGIN
    in.next ← Shorten[cb];
    Output[enableInput, ControlRegister[device.in]];
    END;
  in.last ← cb;
  END;

GetStatus: PUBLIC PROCEDURE [cb: IOCB] RETURNS [status: Status] =
  BEGIN
  RETURN[
    SELECT cb.completion FROM
      0 => pending,
      40000B => ok,
      62000B => tooManyCollisions,
      61000B => packetTooLong,
      70200B => badAlignmentButOkCrc,
      70020B => crc,
      70220B => crcAndBadAlignment,
      70100B, 70120B, 70300B, 70320B => overrun,
      70004B, 70006B => underrun,
      ENDCASE => otherError];
  END;

GetRetries: PUBLIC PROCEDURE [cb: IOCB] RETURNS [CARDINAL] =
  BEGIN
  RETURN[
    SELECT cb.mask FROM
      1 => 0,
      3 => 1,
      7 => 2,
      17B => 3,
      37B => 4,
      77B => 5,
      177B => 6,
      377B => 7,
      777B => 8,
      1777B => 9,
      3777B => 10,
      7777B => 11,
      17777B => 12,
      37777B => 13,
      77777B => 14,
      177777B => 15,
      ENDCASE => 16];
  END;

GetPacketLength: PUBLIC PROCEDURE [cb: IOCB] RETURNS [CARDINAL] =
  BEGIN RETURN[cb.used]; END;
```

```
GetPacketsMissed: PUBLIC PROCEDURE [device: Device] RETURNS [CARDINAL] =
  BEGIN RETURN[ICSBFronDevice[device].missed]; END;

GetNextDevice: PUBLIC PROCEDURE [device: Device] RETURNS [Device] =
  BEGIN OPEN D0InputOutput;
  IF device = nullDeviceHandle THEN
    device ← [0, nullControllerNumber, nullControllerNumber]
  ELSE device.board ← device.board + 1;
  device.in ← GetNextController[ethernetIn, device.in];
  device.out ← GetNextController[ethernetOut, device.out];
  IF device.in = nullControllerNumber OR device.out = nullControllerNumber THEN
    RETURN[nullDeviceHandle];
  RETURN[device];
  END;

TurnOn: PUBLIC PROCEDURE [
  device: Device, host: SpecialSystem.ProcessorID,
  inInterrupt, outInterrupt: WORD, globalState: GlobalStatePtr] =
  BEGIN
  board: Board = device.board;
  out: OCSB = OCSBFronDevice[device];
  in: ICSB = ICSBFronDevice[device];
  StartIO[resetBits[board]];
  out↑ ←
    [next: nilOCB, unused1: 0, unused2: 0, unused3: 0,
      interruptBit: outInterrupt, last: NIL];
  in↑ ←
    [next: nilOCB, host: host, interruptBit: inInterrupt, missed: 0, spare1: 0,
      spare2: 0, buffer: [0, 0, 0, 0], last: NIL];
  StartIO[fixupInputBits[board]];
  StartIO[fixupOutputBits[board]];
  Output[enableInput, ControlRegister[device.in]];
  END;

TurnOff: PUBLIC PROCEDURE [device: Device] =
  BEGIN StartIO[resetBits[device.board]]; END;

-- There is no way to remove a cleanup procedure yet, so we have a flag to avoid duplicates.

alreadyInitializedCleanup: ARRAY Board OF BOOLEAN ← ALL[FALSE];

savedICSB: InputControllerStatusBlock;
savedOCSB: OutputControllerStatusBlock;

AddCleanup: PUBLIC PROCEDURE [device: Device] =
  BEGIN OPEN DeviceCleanup;
  item: Item;
  board: Board = device.board;
  out: OCSB = OCSBFronDevice[device];
  in: ICSB = ICSBFronDevice[device];
  oldHost: SpecialSystem.ProcessorID;
  IF alreadyInitializedCleanup[device.board] THEN RETURN;
  alreadyInitializedCleanup[device.board] ← TRUE;
  DO
    SELECT Await[@item] FROM
      kill => BEGIN StartIO[resetBits[board]]; END;
      turnOff =>
        BEGIN
```



```
        StartIO[resetBits[board]];
        savedICSB ← in↑;
        savedOCSB ← out↑;
        oldHost ← in.host;
        END;
      turnOn =>
        BEGIN
        -- Note that this does NOT really put things back together. It simply smashes things to a s
**afe state. The intention is that the driver will notice that nothing is happening and then call Turn
**Off + TurnOn to reset things. That allows Pilot to reset the GMT clock on the way back from the
**debugger without getting tangled up with the normal Ethernet driver.
        StartIO[resetBits[board]];
        out↑ ←
          [next: nilOCB, unused1: 0, unused2: 0, unused3: 0, interruptBit: 0,
            last: NIL];
        in↑ ←
          [next: nilOCB, host: oldHost,
            -- Ugh, it would be nice if we could do something better
            interruptBit: 0, missed: 0, spare1: 0, spare2: 0,
            buffer: [0, 0, 0, 0], last: NIL];
        StartIO[fixupInputBits[board]];
        StartIO[fixupOutputBits[board]];
        Output[enableInput, ControlRegister[device.in]];
        END;
    ENDCASE;
  ENDLOOP;
END;

RemoveCleanup: PUBLIC PROCEDURE [device: Device] = BEGIN END;

-- Other routines to keep the rest of the world happy

Start: PUBLIC PROCEDURE =
  BEGIN -- Start this module (and rest of chain)
  RemainingStartChain.Start[]
  END;

END. -- EthernetHeadD0.
```

LOG
Time: September 4, 1980 11:02 PM By: HGM, Action: create file.
Time: September 14, 1980 6:33 AM By: HGM, Action: buffer overflow bug.

```
-- EthernetOneHeadD0.mesa (last edited by: HGM on: September 14, 1980 6:37 AM)

DIRECTORY
  Inline USING [DIVMOD, LowHalf],
  Environment USING [Byte],
  HeadStartChain USING [Start],
  DeviceCleanup USING [Item, Await],
  D0InputOutput USING [
    CSB, IOPage, ethernet1In, ethernet1Out, ControllerNumber, GetNextController,
    nullControllerNumber],
  Mopcodes USING [zMISC, zSTARTIO],
  EthernetOneFace;

EthernetOneHeadD0: PROGRAM
  IMPORTS
    D0InputOutput, DeviceCleanup, RemainingStartChain: HeadStartChain, Inline
  EXPORTS EthernetOneFace, HeadStartChain =
  BEGIN OPEN EthernetOneFace;

  -- These are the data structures that the microcode knows about.  If some other module (for exa
**mple, a diagnostic) ever needs this info, it should probably get split out into a separate module.
**For now, it is lumped in here to avoid cluttering up the world with yet another file.

  OCSB: TYPE = LONG POINTER TO OutputControllerStatusBlock;
  OutputControllerStatusBlock: TYPE = MACHINE DEPENDENT RECORD [
    reserved: WORD,
    next: ShortIOCB,
    unused1: WORD,
    unused2: WORD,
    interruptBit: WORD, -- words after here are unused by microcode
    last: IOCB]; -- last IOCB on output queue, valid if next# noIOCB

  ICSB: TYPE = LONG POINTER TO InputControllerStatusBlock;
  InputControllerStatusBlock: TYPE = MACHINE DEPENDENT RECORD [
    reserved: WORD,
    next: ShortIOCB,
    host: LONG CARDINAL,
    interruptBit: WORD,
    missed: WORD, -- for debugging only
    spare1: WORD,
    spare2: WORD,
    buffer: ARRAY [0..4) OF CARDINAL, -- words after here are unused by microcode
    last: IOCB]; -- last IOCB on input queue, valid if next# noIOCB

  IOCB: TYPE = LONG POINTER TO IOControlBlock;
  -- Beware that you don't automatically lengthen one of these.  If you do you will end up with a poi
**nter into your MDS rather then the first 64K where the IOCBs live.  That won't work unless your
**MDS is also the first 64K.
  ShortIOCB: TYPE = POINTER TO IOControlBlock;
  IOControlBlock: TYPE = MACHINE DEPENDENT RECORD [
    next: ShortIOCB,
    mask: WORD,
    spare: WORD,
    completion: WORD,
    used: CARDINAL, -- input only
    length: CARDINAL,
    buffer: LONG POINTER]; -- NB: Must be QuadWord Aligned
```



```
  StartIO: PROCEDURE [SioParameter] = MACHINE CODE BEGIN Mopcodes.zSTARTIO END;

  SioParameter: TYPE = RECORD [WORD];
  firstFixupOutput: SioParameter = [1B];
  firstFixupInput: SioParameter = [2B];
  firstReset: SioParameter = [3B];
  secondFixupOutput: SioParameter = [4B];
  secondFixupInput: SioParameter = [10B];
  secondReset: SioParameter = [14B];

  -- On old boards, output reg 0 of either task is control word for both tasks
  -- On new boards, you must direct the output to the correct half
  -- Input from reg 0 is device id
  -- Input from reg 1 is net&host number switches

  Output: PROCEDURE [Command, Register] = MACHINE CODE
    BEGIN Mopcodes.zMISC, 6; END;

  Input: PROCEDURE [Register] RETURNS [WORD] = MACHINE CODE
    BEGIN Mopcodes.zMISC, 5; END;

  Command: TYPE = RECORD [WORD];
  enableInput: Command = [220B];
  enableOutput: Command = [103B];

  Register: TYPE = MACHINE DEPENDENT RECORD [
    zero: [0..377B] ← 0,
    controller: D0InputOutput.ControllerNumber,
    register: [0..17B]];

  -- completion bits
  processed: WORD = 040000B;
  error: WORD = 020000B;
  hardwareError: WORD = 010000B;
  fragment, tooLong: WORD = 004000B;
  loadOverflow: WORD = 002000B;
  nothingYet: WORD = 0;
  -- 001000 and 000400 are unused so far

  -- Hardware error bits:
  -- 001: Bad Alignment
  -- 002: Bad Parity (between mem and shifter)
  -- 004: Memory Data Fault
  -- 010: CRC
  -- 020: Collision
  -- 040: Input Overrun
  -- 100: Output Overrun
  -- 200: Jam

  noIOCB: ShortIOCB = LOOPHOLE[0];

  Device: TYPE = RECORD [board: Board, in, out: D0InputOutput.ControllerNumber];

  Board: TYPE = [0..2);

  ICSBFronDevice: PROCEDURE [device: Device] RETURNS [ICSB] = INLINE
    BEGIN RETURN[LOOPHOLE[@D0InputOutput.IOPage[device.in]]]; END;
```

```
OCSBFronDevice: PROCEDURE [device: Device] RETURNS [OCSB] = INLINE
  BEGIN RETURN[LOOPHOLE[@D0InputOutput.IOPage[device.out]]]; END;

ControlRegister: PROCEDURE [c: D0InputOutput.ControllerNumber]
  RETURNS [Register] = INLINE
  BEGIN
  RETURN[[0, c, 0]]; -- Register 0 is the control register

  END;

Shorten: PROCEDURE [iocb: IOCB] RETURNS [ShortIOCB] = INLINE
  BEGIN
  -- Maybe we should check to be sure that the high half is zero
  RETURN[Inline.LowHalf[iocb]];
  END;


-- EXPORTed TYPEs

DeviceHandle: PUBLIC TYPE = Device;
ControlBlockRecord: PUBLIC TYPE = IOControlBlock;

-- EXPORTed variables

nullDeviceHandle: PUBLIC DeviceHandle ← LOOPHOLE[123456B];
globalStateSize: PUBLIC CARDINAL ← 0;
controlBlockSize: PUBLIC CARDINAL ← SIZE[IOControlBlock];
hearSelf: PUBLIC BOOLEAN ← TRUE;


-- Non EXPORTed things.  Note that all the state information lives in the CSBs.

fixupInputBits: ARRAY Board OF SioParameter =
  [firstFixupInput, secondFixupInput];
fixupOutputBits: ARRAY Board OF SioParameter =
  [firstFixupOutput, secondFixupOutput];
resetBits: ARRAY Board OF SioParameter = [firstReset, secondReset];

QueueOutput: PUBLIC PROCEDURE [
  device: Device, buffer: LONG POINTER, length: CARDINAL, cb: IOCB] =
  BEGIN
  out: OCSB = OCSBFronDevice[device];
  cb↑ ←
    [next: nolOCB, mask: 0, spare: 0, completion: 0, used: 0, length: length,
      buffer: buffer];
  IF out.next = nolOCB THEN
    BEGIN -- new iocb, hardware idle
    out.next ← Shorten[cb];
    Output[enableOutput, ControlRegister[device.out]]; -- poke hardware

    END
  ELSE
    BEGIN -- output active, add to end of chain
    out.last.next ← Shorten[cb];
    IF out.next = nolOCB AND cb.completion = 0 THEN
      BEGIN -- oops, hardware went idle
      out.next ← Shorten[cb];
      Output[enableOutput, ControlRegister[device.out]]; -- poke hardware
```



```
      END;
    END;
  out.last ← cb;
  END;

QueueInput: PUBLIC PROCEDURE [
  device: Device, buffer: LONG POINTER, length: CARDINAL, cb: IOCB] =
  BEGIN
  in: ICSB = ICSBFronDevice[device];
  cb↑ ←
    [next: nolOCB, mask: 0, spare: 0, completion: 0, used: 0, length: length,
      buffer: buffer];
  IF in.next # nolOCB THEN in.last.next ← Shorten[cb];
  IF in.next = nolOCB AND cb.completion = 0 THEN
    BEGIN
    in.next ← Shorten[cb];
    -- Poke the microcode so it will notice this buffer, delete the next line when the microcode is f
**ixed
    StartIO[fixupInputBits[device.board]];
    Output[enableInput, ControlRegister[device.in]];
    END;
  in.last ← cb;
  END;

GetStatus: PUBLIC PROCEDURE [cb: IOCB] RETURNS [status: Status] =
  BEGIN
  RETURN[
    SELECT cb.completion FROM
      0 => pending,
      40000B => ok,
      62000B => tooManyCollisions,
      61000B => packetTooLong,
      70001B => badAlignmentButOkCrc,
      70010B => crc,
      70011B => crcAndBadAlignment,
      70040B, 70041B, 70050B, 70051B => overrun,
      70100B, 70120B => underrun,
      ENDCASE => otherError];
  END;

GetRetries: PUBLIC PROCEDURE [cb: IOCB] RETURNS [CARDINAL] =
  BEGIN
  RETURN[
    SELECT cb.mask FROM
      1 => 0,
      3 => 1,
      7 => 2,
      17B => 3,
      37B => 4,
      77B => 5,
      177B => 6,
      377B => 7,
      777B => 8,
      1777B => 9,
      3777B => 10,
      7777B => 11,
      17777B => 12,
      37777B => 13,
```

```
    77777B => 14,
    177777B => 15,
    ENDCASE => 16];
  END;

GetPacketLength: PUBLIC PROCEDURE [cb: IOCB] RETURNS [CARDINAL] =
  BEGIN RETURN[cb.used]; END;

GetPacketsMissed: PUBLIC PROCEDURE [device: Device] RETURNS [CARDINAL] =
  BEGIN RETURN[ICSBFronDevice[device].missed]; END;

GetNextDevice: PUBLIC PROCEDURE [device: Device] RETURNS [Device] =
  BEGIN OPEN D0InputOutput;
  IF device = nullDeviceHandle THEN
    device ← [0, nullControllerNumber, nullControllerNumber]
  ELSE device.board ← device.board + 1;
  device.in ← GetNextController[ethernet1In, device.in];
  device.out ← GetNextController[ethernet1Out, device.out];
  IF device.in = nullControllerNumber OR device.out = nullControllerNumber THEN
    RETURN[nullDeviceHandle];
  RETURN[device];
  END;

GetEthernet1Address: PUBLIC PROCEDURE [device: Device]
  RETURNS [net, host: Environment.Byte] =
  BEGIN
  reg: Register = [controller: device.in, register: 1];
  [net, host] ← Inline.DIVMOD[Input[reg], 400B];
  END;

TurnOn: PUBLIC PROCEDURE [
  device: Device, host: Environment.Byte, inInterrupt, outInterrupt: WORD,
  globalState: GlobalStatePtr] =
  BEGIN
  board: Board = device.board;
  out: OCSB = OCSBFronDevice[device];
  in: ICSB = ICSBFronDevice[device];
  StartIO[resetBits[board]];
  out↑ ←
    [reserved: 0, next: nolOCB, unused1: 0, unused2: 0,
      interruptBit: outInterrupt, last: NIL];
  in↑ ←
    [reserved: 0, next: nolOCB, host: host, interruptBit: inInterrupt,
      missed: 0, spare1: 0, spare2: 0, buffer: [0, 0, 0, 0], last: NIL];
  StartIO[fixupInputBits[board]];
  StartIO[fixupOutputBits[board]];
  Output[enableInput, ControlRegister[device.in]];
  END;

TurnOff: PUBLIC PROCEDURE [device: Device] =
  BEGIN StartIO[resetBits[device.board]]; END;

-- There is no way to remove a cleanup procedure yet, so we have a flag to avoid duplicates.

alreadyInitializedCleanup: ARRAY Board OF BOOLEAN ← ALL[FALSE];

savedICSB: InputControllerStatusBlock;
savedOCSB: OutputControllerStatusBlock;
```



```
AddCleanup: PUBLIC PROCEDURE [device: Device] =
  BEGIN OPEN DeviceCleanup;
  item: Item;
  board: Board = device.board;
  out: OCSB = OCSBFronDevice[device];
  in: ICSB = ICSBFronDevice[device];
  oldHost: LONG CARDINAL;
  IF alreadyInitializedCleanup[device.board] THEN RETURN;
  alreadyInitializedCleanup[device.board] ← TRUE;
  DO
    SELECT Await[@item] FROM
      kill => BEGIN StartIO[resetBits[board]]; END;
      turnOff =>
        BEGIN
        StartIO[resetBits[board]];
        savedICSB ← in↑;
        savedOCSB ← out↑;
        oldHost ← in.host;
        END;
      turnOn =>
        BEGIN
        -- Note that this does NOT really put things back together.  It simply smashes things to a s
**afe state.  The intention is that the driver will notice that nothing is happening and then call Turn
**Off + TurnOn to reset things.  That allows Pilot to reset the GMT clock on the way back from the
**debugger without getting tangled up with the normal Ethernet driver.
        StartIO[resetBits[board]];
        out↑ ←
          [reserved: 0, next: nolOCB, unused1: 0, unused2: 0, interruptBit: 0,
            last: NIL];
        in↑ ←
          [reserved: 0, next: nolOCB, host: oldHost,
            -- Ugh, it would be nice if we could do something better
            interruptBit: 0, missed: 0, spare1: 0, spare2: 0,
            buffer: [0, 0, 0, 0], last: NIL];
        StartIO[fixupInputBits[board]];
        StartIO[fixupOutputBits[board]];
        Output[enableInput, ControlRegister[device.in]];
        END;
      ENDCASE;
    ENDLOOP;
  END;

RemoveCleanup: PUBLIC PROCEDURE [device: Device] = BEGIN END;

-- Other routines to keep the rest of the world happy

Start: PUBLIC PROCEDURE =
  BEGIN -- Start this module (and rest of chain)
  RemainingStartChain.Start[]
  END;

END. -- EthernetOneHeadD0.
```

LOG
Time: August 20, 1980 2:01 PM By: BLyon, Action: renamed EthernetFace and EthernetHeadD0 to
***EthernetOneFace and EthernetOneHeadD0, resp.*
Time: August 25, 1980 5:36 PM By: BLyon, Action: added [ELSE h.board ← h.board + 1] clause to
***GetNextDevice.*

Time: September 4, 1980 11:02 PM By: HGM, Action: add hearSelf, use exported types.
Time: September 14, 1980 6:36 AM By: HGM, Action: bug in bufferOverflow.

```
-- GMTUsingIntervalTimer.mesa (last edited by: McJones on: June 26, 1980  10:48 AM)

DIRECTORY
  DeviceCleanup USING [Perform],
  HeadStartChain USING [Start],
  PilotMP USING [cPowerOff], -- heads should have their own MP codes!
  ProcessInternal USING [DisableInterrupts],
  ProcessorFace USING [
    BootButton, GetClockPulses, gmtEpoch, GreenwichMeanTime,
    microsecondsPerHundredPulses, SetMP];

GMTUsingIntervalTimer: PROGRAM
  IMPORTS
    DeviceCleanup, RemainingHeads: HeadStartChain, ProcessInternal, ProcessorFace
  EXPORTS HeadStartChain, ProcessorFace =
  BEGIN OPEN ProcessorFace;
  --
  -- Simulate greenwich mean time and power control features of ProcessorFace using interval tim
**er
  -- Greenwich mean time
  GetGreenwichMeanTime: PUBLIC PROCEDURE RETURNS [GreenwichMeanTime] =
    BEGIN
    IF gmtSimulated ~= gmtEpoch THEN -- don't update clock unless it has been set
      BEGIN
      seconds: GreenwichMeanTime =
        (GetClockPulses[] - pulsesGmtSimulated)/pulsesPerSecond;
      pulsesGmtSimulated ← pulsesGmtSimulated + seconds*pulsesPerSecond;
      -- long multiply!
      gmtSimulated ← gmtSimulated + seconds;
      END;
    RETURN[gmtSimulated]
    END;

  SetGreenwichMeanTime: PUBLIC PROCEDURE [gmt: GreenwichMeanTime] =
    BEGIN pulsesGmtSimulated ← GetClockPulses[]; gmtSimulated ← gmt; END;

  gmtSimulated: GreenwichMeanTime ← gmtEpoch; -- = gmtEpoch => not set
  pulsesGmtSimulated: LONG CARDINAL;
  -- interval timer value corresponding to gmtSimulated
  pulsesPerSecond: LONG CARDINAL = 1D6*100/microsecondsPerHundredPulses;
  PowerOff: PUBLIC PROCEDURE =
    BEGIN
    -- NOTE: This code depends on the greenwich mean time clock running with interrupts disable
**d and devices turned off.
    ProcessInternal.DisableInterrupts[];
    DeviceCleanup.Perform[turnOff];
    SetMP[PilotMP.cPowerOff];
    DO
      -- forever
      IF GetGreenwichMeanTime[] - gmtEpoch >= gmtAutomaticPowerOn - gmtEpoch AND
        (~externalEventRequired OR ExternalEvent[]) THEN BootButton[]
      ENDLOOP
    END;

  gmtAutomaticPowerOn: GreenwichMeanTime;
  externalEventRequired: BOOLEAN;
  ExternalEvent: PROCEDURE RETURNS [BOOLEAN] = BEGIN RETURN[FALSE] END;

  SetAutomaticPowerOn: PUBLIC PROCEDURE [
```



```
    gmt: GreenwichMeanTime, externalEvent: BOOLEAN] =
    BEGIN gmtAutomaticPowerOn ← gmt; externalEventRequired ← externalEvent; END;

  ResetAutomaticPowerOn: PUBLIC PROCEDURE =
    BEGIN
    gmtAutomaticPowerOn ← gmtEpoch - 1; -- infinity

    END;
  --
  -- (Head)StartChain

  Start: PUBLIC PROCEDURE =
    -- Start this module (and rest of chain)
    BEGIN RemainingHeads.Start[] END;
  --
  -- Initialization

  ResetAutomaticPowerOn[];
  END.
LOG
Time: February 4, 1980  10:36 AM          By: McJones          Action: Create file, borrowing cod
**e from SystemImpl
Time: June 26, 1980  10:48 AM  By: McJones          Action: OISProcessorFace = >ProcessorFac
**e
```

```
-- ProcessorHeadD0.mesa (last edited by: Forrest on: October 3, 1980 5:18 PM)

DIRECTORY
  D0InputOutput USING [
    ControllerNumber, ControllerType, CSBArray, Input, null, nullControllerNumber,
    rdc, rdcWithServiceLate],
  Environment USING [Byte, PageCount, PageNumber, wordsPerPage],
  Frame USING [GetReturnLink],
  Inline USING [BITOR],
  MiscAlpha USING [aLOADRAMJ, aREADRAM, aTEXTBLT],
  Mopcodes USING [zMISC],
  PrincOps USING [ControlLink, FrameHandle, StateVector],
  ProcessorFace USING [ProcessorID],
  SDDefs USING [SD, sUnimplemented];

ProcessorHeadD0: PROGRAM
  IMPORTS Frame, D0InputOutput, Inline
  EXPORTS D0InputOutput, ProcessorFace
  SHARES ProcessorFace = PUBLIC

  BEGIN OPEN D0InputOutput;

  --
  -- D0InputOutput

  IOPage: LONG POINTER TO CSBArray ← LOOPHOLE[LONG[
    pageIO*Environment.wordsPerPage]];
  firstController: ControllerNumber. = 4B; -- tasks 0B thru 3B taken up by emulator
  lastController: ControllerNumber = 15B;
  -- tasks 16B and 17B taken by timers and faults
  ControllerIndex: TYPE = ControllerNumber [firstController..lastController];
  controllerRoster: ARRAY ControllerIndex OF ControllerType ← ALL[null];

  GetNextController: PROC [type: ControllerType, prev: ControllerNumber]
    RETURNS [ControllerNumber] =
    BEGIN
    i: ControllerIndex;
    FOR i DECREASING IN ControllerIndex DO
      IF type = controllerRoster[i] AND (prev = nullControllerNumber OR prev > i)
        THEN RETURN[i];
      ENDLOOP;
    RETURN[nullControllerNumber]
    END;

  InitializeD0InputOutput: PRIVATE PROC =
    BEGIN
    IDRegister: TYPE = MACHINE DEPENDENT RECORD [
      type: ControllerType, lowByte: [0..256)];
    r: IDRegister; -- code generator bug in 6.0n requires this
    idRegNo: [0..17B] = 0;
    i: ControllerIndex;
    t: ControllerType;
    -- Initialization microcode really should have put roster into memory somewhere.
    FOR i IN ControllerIndex DO
      r ← LOOPHOLE[Input[[controller: i, register: idRegNo]], IDRegister];
      t ← r.type;
      IF t = rdcWithServiceLate THEN t ← rdc; -- #*%$@!! RDC
      controllerRoster[i] ← t;
```



```
    ENDLOOP;
    END;

  --
  -- ProcessorFace

  -- Initialization

  Start: PROC = { -- By now start trap has occurred and main-body code has run--};

  -- Processor id

  processorID: ProcessorFace.ProcessorID ← GetProcessorID[];
  GetProcessorID: PRIVATE PROC RETURNS [ProcessorFace.ProcessorID] =
    BEGIN
    csw: CSWord = ReadRam[7777B];
    CSWord: TYPE = MACHINE DEPENDENT RECORD [
      bits16To31: CARDINAL,
      bits0To15: CARDINAL,
      bits32To35: [0..17B],
      address: [0..7777B]];
    ReadRam: PROC [address: [0..7777B]] RETURNS [CSWord] = MACHINE CODE
      BEGIN Mopcodes.zMISC, MiscAlpha.aREADRAM END;
    RETURN[[0, csw.bits0To15, csw.bits16To31]] -- note most significant first

    END;

  -- Real memory configuration

  dedicatedRealMemory: Environment.PageCount ← 0;

  -- Virtual memory layout
  GetNextAvailableVM: PROC [page: Environment.PageNumber]
    RETURNS [firstPage: Environment.PageNumber, count: Environment.PageCount] =
    BEGIN
    IF page < page1 THEN RETURN[page, page1 - page];
    page ← MAX[page, page1 + 1];
    IF page < page376B THEN RETURN[page, page376B - page];
    page ← MAX[page, page376B + 1];
    IF page < pageIO THEN RETURN[page, pageIO - page];
    page ← MAX[page, pageIO + 1];
    IF page < pageFirstUnimplemented THEN
      RETURN[page, pageFirstUnimplemented - page];
    RETURN[LAST[Environment.PageNumber], 0];
    END;

  page1: PRIVATE Environment.PageNumber = 1B;
  -- I/O page for Alto-compatible Diablo31, display, cursor, mouse coords
  page376B: PRIVATE Environment.PageNumber = 376B;
  -- I/O page for Alto-compatible keyboard/mouse bitmap
  pageIO: PRIVATE Environment.PageNumber = 377B; -- "official" D0 I/O page
  pageFirstUnimplemented: PRIVATE Environment.PageNumber = 40000B;
  -- [pageFirstUnimplemented..LAST[PageNumber]] not implemented

  -- Interval time
  microsecondsPerHundredPulses: CARDINAL ← 64*100;

  -- Naked notifies
```

```
cvTimeoutMask: PRIVATE WORD = 100000B; -- D0 microcode assumes level 0
reservedNakedNotifyMask: WORD ← cvTimeoutMask;

-- Condition variable time
-- millisecondsPerTick is exported by UserTerminalHeadD0,
--  since its value depends on the display refresh rate.
InitializeCVTimeouts: PRIVATE PROC =
  BEGIN
  DIW: LONG POINTER TO WORD = LOOPHOLE[LONG[421B]];
  DIW↑ ← Inline.BITOR[DIW↑, cvTimeoutMask];
  END;

-- Booting and power control

BootButton: PROC =
  BEGIN
  LoadRam: PROC [pMicrocode: LONG POINTER, andJump: BOOLEAN] = MACHINE CODE
    BEGIN Mopcodes.zMISC, MiscAlpha.aLOADRAMJ END;
  bootMe: ARRAY [0..7) OF WORD ←
    [000000B, 170000B, 000047B, 015000B, 177760B, 163351B, 007400B];
  LoadRam[@bootMe, TRUE] -- never returns

  END;

UnimplementedInstruction: ERROR = CODE; -- This should actually be in traps

-- Software assist for microcode
-- assumes return link is a frame, and has been started
UnimplementedTrap: PRIVATE PROC =
  BEGIN
  BumpPC: PROC [f: PrincOps.FrameHandle, i: INTEGER] = INLINE {
    f.pc ← [f.pc + i]};
  l: PrincOps.ControlLink;
  code: LONG POINTER TO PACKED ARRAY [0..0) OF Environment.Byte;
  state: PrincOps.StateVector;

  state ← STATE;
  state.source ← l ← Frame.GetReturnLink[];
  -- WHILE ~l.proc AND l.indirect DO l ← l.link↑ ENDLOOP;
  -- IF l.proc THEN ERROR;
  code ← LOOPHOLE[l.frame.accesslink.code.longbase];
  IF code[l.frame.pc] = Mopcodes.zMISC THEN
    SELECT code[l.frame.pc + 1] FROM
      MiscAlpha.aTEXTBLT =>
        BEGIN
        state.dest ← LOOPHOLE[ --SoftwareTextBlt.--TEXTBLT,
          PrincOps.ControlLink];
        BumpPC[l.frame, 2];
        RETURN WITH state;
        END;
      ENDCASE;
  ERROR UnimplementedInstruction
  END;

TEXTBLT: PRIVATE PROC = {ERROR UnimplementedInstruction};

--
-- Initialization
```



```
  InitializeCVTimeouts[];
  InitializeD0InputOutput[];
  SDDefs.SD[SDDefs.sUnimplemented] ← UnimplementedTrap;

  END.
```

August 8, 1979 1:47 AM	*Redell*	*Create file*
October 17, 1979 6:54 PM	*Redell*	*Add patch to detect RDC thinking it's an old ethernet*
February 4, 1980 10:36 AM	*McJones*	*Rename OISProcessorFaceD0Impl = >OISProcessorHe*
***adD0; delete device registry/enumeration; add new OISProcessorFace items*		
February 25, 1980 3:06 PM	*McJones*	*Move export of millisecondsPerTick to UserTerminalHe*
***adD0*		
April 30, 1980 10:17 AM	*Forrest*	*Add InitializeCVTimeout*
May 14, 1980 4:27 PM	*McJones*	*Add reservedNakedNotifyMask; remove Start from Hea*
***dStartChain (export to OISProcessorFace now); change GetNextController to deliver highest co*		
***ntroller number first*		
June 25, 1980 4:23 PM	*McJones*	*Drop "OIS" from name; expand processor id to 48 bits*
August 12, 1980 10:26 AM	*McJones*	*Add dedicatedRealMemory; use MiscAlpha*
August 26, 1980 11:17 AM	*McJones*	*Change cvTimeoutMask for new process microcode*
October 3, 1980 5:17 PM	*Forrest*	*Move in unimplemented trap*

```
-- SA4000HeadD0.mesa (last edited by: McJones on: July 24, 1980 11:23 AM)
-- Things to do:
-- 1) GetNextDevice should find all drives on a controller (does this need a microcode change?)
-- 2) The semantics of aborting should be independent of controllers; e.g. aborting could work on
**all drives, or on an individual drive. (Currently it works on all drives of a controller, but it is hard
**to see how this can be specified in a controller-independent way.)

DIRECTORY
  DeviceCleanup USING [Await, Item, Reason],
  D0InputOutput USING [
    ControllerNumber, GetNextController, IOPage, nullControllerNumber, rdc],
  Environment USING [Base, first64K, Long],
  HeadStartChain USING [Start],
  Inline USING [BITAND, LowHalf],
  PilotDisk USING [Handle, Label],
  PilotFileTypes USING [PilotVFileType],
  RDC,
  SA4000Face,
  System USING [],
  SystemInternal USING [UniversalID];

SA4000HeadD0: PROGRAM
  IMPORTS
    D0InputOutput, DeviceCleanup, RemainingHeads: HeadStartChain, Inline, RDC
  EXPORTS HeadStartChain, SA4000Face, System
  SHARES SA4000Face =
  BEGIN OPEN RDC, SA4000Face;
  -- TYPE DEFINITIONS
  CSB: TYPE = MACHINE DEPENDENT RECORD [
    -- Controller status block. There is one CSB for each controller, with a maximum of four contro
**llers. The CSB resides in a resident page of memory. D0InputOutput.IOPage is a LONG POINTE
**R to an array of CSBs. The CSB is indexed by the controller number in the DiskHandle.
    cylinder: ARRAY Drive OF CARDINAL,
    -- Contains the current cylinder address for each drive or -1 if the drive is to be recalibrated bef
**ore the next command.
    state: CSBState];
  CSBState: TYPE = MACHINE DEPENDENT RECORD [
    -- This part of the CSB is defined separately so that it can be saved and restored by InitializeCle
**anup.
    next: IOCBshortPtr,
    -- Points to the next IOCB to be processed by the microcode. Points to the last IOCB if an error
**is reported by the microcode. Contains zero if the queue is empty. This is updated by Initiate an
**d by the microcode. When the microcode finishes processing an IOCB, it replaces next with the
**next pointer from the IOCB.
    deferring: INTEGER,
    -- This is set to -1 by the microcode when it reports an error completion to the Head. The micro
**code will not process any more IOCBs until this is set to zero again by Poll when the error is rep
**orted to the client. In the case of a labelCheck that requires a fixup, the error is not really an err
**or at all, and the client will be unaware of the label fixup.
    tail: IOCBshortPtr,
    -- Points to the last IOCB in the queue. Used only by Initiate.
    transferMask: WORD];
  -- This interupt mask is used by the microcode to schedule Mesa processes at the completion of
**each IOCB. It is initialized by StartB.
  CSBPtr: TYPE = LONG POINTER TO CSB;
  DiskHandle: TYPE = MACHINE DEPENDENT RECORD [
    -- representation of DeviceHandle
    zero: [0..16) ← 0,
```



```
    controller: D0InputOutput.ControllerNumber,
    drive: [0..256)];
  Drive: TYPE = [0..drives);
  -- CONSTANTS
  drives: CARDINAL = 4; -- maximum number of drives per controller
  rdcCommands: ARRAY Command OF WORD =
    [
      -- These are the RDC operation codes that correspond to the LabelDataOperations. For each
**operation, header verify is assumed. Each command includes the allow wake bit (4000).
      4110B, -- headerVerifyLabelVerify
      4112B, -- headerVerifyLabelVerifyDataRead
      4114B, -- headerVerifyLabelVerifyDataWrite
      4111B, -- headerVerifyLabelVerifyDataVerify
      4140B, -- headerVerifyLabelWrite
      4144B, -- headerVerifyLabelWriteDataWrite
      4120B, -- headerVerifyLabelRead
      4122B, -- headerVerifyLabelReadDataRead
      4124B, -- headerVerifyLabelReadDataWrite
      4121B, -- headerVerifyLabelReadDataVerify
      4510B, -- headerReadLabelVerify
      4512B, -- headerReadLabelVerifyDataRead
      4514B, -- headerReadLabelVerifyDataWrite
      4511B, -- headerReadLabelVerifyDataVerify
      4540B, -- headerReadLabelWrite
      4544B, -- headerReadLabelWriteDataWrite
      4520B, -- headerReadLabelRead
      4522B, -- headerReadLabelReadDataRead
      4524B, -- headerReadLabelReadDataWrite
      4521B, -- headerReadLabelReadDataVerify
      4200B]; -- headerWrite
  labelRead: WORD = 0020B; -- bit of RDC command signifying label read
  p: Environment.Base = Environment.first64K;
  nil: Environment.Base RELATIVE POINTER = LOOPHOLE[0];
  nullCylinder: CARDINAL = 177777B; -- used for initialization and recalibration
  nullDiskHandle: DiskHandle =
    [controller: D0InputOutput.nullControllerNumber, drive: 0];
  -- GetNextDevice depends on this value
  rdcInProgress: WORD = 0;
  serviceLateRetries: CARDINAL = 1000; -- retries to be performed by the microcode
  rateErrorRetries: CARDINAL = 1000; -- retries to be performed by the microcode
  -- GLOBAL VARIABLES
  globalStateSize: PUBLIC CARDINAL ← 0;
  -- an IOCB for label fixup is not required for the D0 implementation.
  nullDeviceHandle: PUBLIC DeviceHandle ← Seal[nullDiskHandle];
  operationSize: PUBLIC CARDINAL ← SIZE[IOCB];
  totalErrors: CARDINAL ← 0; -- = total errors reported by Poll
  uCodeServiceLateRetries: CARDINAL ← 0;
  -- total serviceLate retries performed by the microcode
  uCodeRateErrorRetries: CARDINAL ← 0;
  -- total rateError retries performed by the microcode
  -- PROCEDURES
  GetDeviceAttributes: PUBLIC PROCEDURE [device: DeviceHandle]
    RETURNS [cylinders, movingHeads, fixedHeads, sectorsPerTrack: CARDINAL] =
    -- MUST BE MODIFIED TO HANDLE FIXED HEADS, SINGLE-PLATTER DRIVES
    BEGIN
    RETURN[
      cylinders: 202, movingHeads: 8, -- actually depends on whether 4004 or 4008
        fixedHeads: 0, -- actually depends on option
```

```
    sectorsPerTrack: 28]
  END;

GetNextDevice: PUBLIC PROCEDURE [device: DeviceHandle] RETURNS [DeviceHandle] =
  -- MUST BE MODIFIED TO HANDLE MULTIPLE DRIVES/CONTROLLER
  BEGIN OPEN DOInputOutput, disk: LOOPHOLE[device, DiskHandle];
  controller: ControllerNumber;
  SELECT disk FROM
    nullDiskHandle =>
      BEGIN
      controller ← GetNextController[rdc, nullControllerNumber];
      IF controller = nullControllerNumber THEN GOTO Null; -- no drives
      RETURN[Seal[[controller: controller, drive: 0]]]
      END;
    ENDCASE => GOTO Null
  EXITS Null => RETURN[nullDeviceHandle];
  END;

Initialize: PUBLIC PROCEDURE [t: WORD, globalState: GlobalStatePtr] =
  BEGIN OPEN DOInputOutput;
  csbState: CSBState = [next: nil, deferring: 0, tail: nil, transferMask: t];
  c: ControllerNumber ← nullControllerNumber;
  WHILE (c ← GetNextController[rdc, c]) ~= nullControllerNumber DO
    OPEN csb: LOOPHOLE[@IOPage[c], CSBPtr]; csb.state ← csbState; ENDLOOP;
  END;

InitializeCleanup: PUBLIC PROCEDURE =
  BEGIN OPEN DOInputOutput;
  c: ControllerNumber ← nullControllerNumber;
  WHILE (c ← GetNextController[rdc, c]) ~= nullControllerNumber DO
    OPEN csb: LOOPHOLE[@IOPage[c], CSBPtr];
    InitializeCleanupForController[@csb];
    ENDLOOP;
  END;

Initiate: PUBLIC PROCEDURE [operationPtr: OperationPtr] =
  BEGIN
  -- This procedure initiates an operation. The client passes an IOCB with some of the parameter
**s filled in. We fill in the parameters that are unique to this implementation of the Head and micro
**code. Then we chain the IOCB onto the CSB. The microcode wakes up at every sector to see if t
**here is an IOCB waiting in the CSB queue.
  iocb: IOCBlongPtr = LOOPHOLE[operationPtr];
  iocbShort: IOCBshortPtr = LOOPHOLE[Inline.LowHalf[operationPtr]];
  csb: CSBPtr = IGetCSB[iocb.operation.device];
  -- Initialize clientHeader, clientLabel, status, RDC command, and next fields.
  -- Design of RDC dictates position of clientHeader and (8 word) clientLabel fields in IOCB.
  iocb.clientHeader ← iocb.operation.clientHeader;
  PackLabel[to: @iocb.clientLabel, from: iocb.operation.labelPtr];
  iocb.operation.deviceStatus.a ← rdcInProgress;
  iocb.rdcCommand ← rdcCommands[iocb.operation.command];
  iocb.next ← nil;
  -- The microcode performs error retries automatically for serviceLate and rateError because th
**ey occur frequently.
  iocb.serviceLateRetryCount ← serviceLateRetries;
  iocb.rateErrorRetryCount ← rateErrorRetries;
  -- Chain this IOCB onto CSB for processing by microcode.
  IF csb.state.next = nil THEN
    -- microcode is currently not busy, so chain next IOCB onto head of list.
```



```
    csb.state.next ← iocbShort
  ELSE
    -- microcode is busy processing an IOCB, but it could report completion at any time. Chain n
**ew IOCB onto tail of queue. There is a possible race, which is handled in Poll.
    p[csb.state.tail].next ← iocbShort;
  csb.state.tail ← iocbShort;
  END;

Poll: PUBLIC PROCEDURE [operationPtr: OperationPtr] RETURNS [status: Status] =
  BEGIN
  iocb: IOCBlongPtr = LOOPHOLE[operationPtr];
  csb: CSBPtr = IGetCSB[iocb.operation.device];
  controllerIdle: BOOLEAN = csb.state.next = nil OR csb.state.deferring < 0;
  -- used to detect race noted in Initiate
  -- Update the error counts performed by the microcode. The number of retries in the IOCB will
**be decremented by one for each retry. We keep track of these for debugging purposes.
  uCodeServiceLateRetries ←
    uCodeServiceLateRetries + (serviceLateRetries - iocb.serviceLateRetryCount);
  uCodeRateErrorRetries ←
    uCodeRateErrorRetries + (rateErrorRetries - iocb.rateErrorRetryCount);
  status ← DecodeStatus[iocb.operation.deviceStatus.a];
  -- check status of specified IOCB
  IF status = inProgress THEN
    -- operation not completed (possibly not initiated)
    -- If controller went idle before noticing this request, resubmit it (and following IOCBs).
    BEGIN
    IF controllerIdle THEN
      csb.state.next ← LOOPHOLE[Inline.LowHalf[operationPtr]];
    RETURN
    END;
  IF status = goodCompletion THEN NULL -- operation successfully completed

  ELSE -- operation unsuccessfully completed
    BEGIN
    -- For all of the following completion codes, the microcode has reported an error. csb.next sti
**ll points to the IOCB that caused the error. csb.deferring has been set to -1 by the microcode to
**prevent it from processing any more IOCBs until we set it back to zero.
    SELECT status FROM
      labelCheck =>
        BEGIN
        -- The RDC microcode has read the actual label from the disk into the diskLabel of the IO
**CB. The RDC verifies all 8 words of the label, but we are only interested in verifying the "signific
**ant" ones.
        IF MatchLabels[@iocb.clientLabel, @iocb.diskLabel] THEN
          -- not an error after all
          BEGIN
          -- If all the label words that we want to verify match the label words on the disk, we need
**only resubmit the IOCB again. First we update the clientHeader field in the operation to match t
**he one in the IOCB which has been incremented by the microcode and we update the bootChai
**nLink field in the operation label to match the corresponding field in the disk label.
          iocb.operation.clientHeader ← iocb.clientHeader;
          iocb.clientLabel.bootChainLink ← iocb.diskLabel.bootChainLink;
          iocb.operation.deviceStatus.a ← rdcInProgress;
          csb.state.deferring ← 0; -- restart the controller
          RETURN[inProgress]
          END;
        END;
      wrongSector =>
        -- The code that was reported by the Controller is headerError. This means that the heade
```

```
**r read from the disk did not match the header sent by the client. It could be a wrongCylinder, wr
**ongHeader, or wrongSector.
        SELECT TRUE FROM
          iocb.operation.clientHeader.cylinder ~=
            iocb.operation.diskHeader.cylinder => status ← wrongCylinder;
          iocb.operation.clientHeader.head ~= iocb.operation.diskHeader.head =>
            status ← wrongHead;
          ENDCASE;
      -- wrongSector corresponds to headerError, so we don't need to change status.

      ENDCASE; -- other error
    totalErrors ← totalErrors + 1;
    csb.state.next ← nil; -- zap all deferred requests
    csb.state.deferring ← 0; -- restart the controller

    END;
  iocb.operation.clientHeader ← iocb.clientHeader;
  UnpackLabel[
    to: iocb.operation.labelPtr,
    from:
    IF Inline.BITAND[iocb.rdcCommand, labelRead] ~= 0 THEN @iocb.diskLabel
    ELSE @iocb.clientLabel];
  END;

Recalibrate: PUBLIC PROCEDURE [device: DeviceHandle] =
  BEGIN
  GetCSB[device].cylinder[LOOPHOLE[device, DiskHandle].drive] ← nullCylinder
  END;

Reset: PUBLIC PROCEDURE [device: DeviceHandle] =
  BEGIN
  -- WHAT SHOULD THIS DO?

  END;

Start: PUBLIC PROCEDURE = -- exported to HeadStartChain
  BEGIN RemainingHeads.Start[]; END;
-- Private procedures

DecodeStatus: PROCEDURE [r: UNSPECIFIED] RETURNS [s: Status] =
  -- decode RDC status word
  INLINE
  BEGIN
  RETURN[LOOPHOLE[LOOPHOLE[r, RDC.DeviceStatus].outcome]];
  -- completion code in processor field matches Status codes up to wrongSector

  END;

GetCSB: PROCEDURE [device: DeviceHandle] RETURNS [CSBPtr] =
  BEGIN RETURN[IGetCSB[device]] END;

IGetCSB: PROCEDURE [device: DeviceHandle] RETURNS [CSBPtr] = INLINE
  BEGIN
  RETURN[
    LOOPHOLE[@D0InputOutput.IOPage[LOOPHOLE[device, DiskHandle].controller]]]
  END;

InitializeCleanupForController: PROCEDURE [csb: CSBPtr] =
  BEGIN OPEN DeviceCleanup;
```



```
  item: Item;
  reason: Reason;
  csbState: CSBState;
  DO
    reason ← Await[@item];
    SELECT reason FROM
      turnOff, kill =>
        BEGIN
        UNTIL csb.state.next = nil OR csb.state.deferring < 0 DO ENDLOOP;
        csbState ← csb.state; -- save state of CSB

        END;
      turnOn => csb.state ← csbState; -- restore CSB state

      ENDCASE
    ENDLOOP
  END;

MatchLabels: PROCEDURE [p1, p2: LONG POINTER TO RDC.PackedLabel]
  RETURNS [BOOLEAN] =
  -- Like PilotDisk.MatchLabel, for packed labels.
  -- All fields but bootChainLink are significant.
  -- (Really should treat filePageHi as significant when type IN PilotVolumeFileType.)
  BEGIN
  MatchableLabel: TYPE = ARRAY [0..8) OF RECORD [UNSPECIFIED];
  FOR i: CARDINAL IN [0..7) DO
    IF LOOPHOLE[p1↑, MatchableLabel][i] ~= LOOPHOLE[p2↑, MatchableLabel][i] THEN
      RETURN[FALSE];
    ENDLOOP;
  RETURN[TRUE]
  END;

UniversalID: PUBLIC TYPE = SystemInternal.UniversalID;
PackLabel: PROCEDURE [
  to: LONG POINTER TO RDC.PackedLabel, from: LONG POINTER TO PilotDisk.Label] =
  INLINE
  BEGIN
  OPEN fileID: LOOPHOLE[from.fileID, SystemInternal.UniversalID],
    -- Mesa AR 4669
    seq: LOOPHOLE[ --from.--fileID.sequence, Environment.Long],
    s: LOOPHOLE[seq.highbits, MACHINE DEPENDENT RECORD [
      bits0Thru1(0:0..1): CARDINAL [0..4),
      bit2(0:2..2): CARDINAL [0..2),
      bits3Thru15(0:3..15): CARDINAL [0..8192)]];
  to↑ ←
    [sequence0Bit2: s.bit2, processorID0: --from.--fileID.processor.a,
      processorID1: --from.--fileID.processor.b,
      processorID2: --from.--fileID.processor.c,
      sequence1: Inline.LowHalf[ --from.--fileID.sequence],
      sequence0Bits3Thru15: s.bits3Thru15, immutable: from.immutable,
      temporary: from.temporary, zeroSize: from.zeroSize,
      filePageLo: from.filePageLo, type: from.type, vp: NULL];
  IF from.type IN PilotFileTypes.PilotVFileType THEN
    to.vp ← volumeFile[filePageHi: from.filePageHi]
  ELSE to.vp ← normalFile[PackDiskAddress[LOOPHOLE[from.bootChainLink]]];
  END;

Seal: PROCEDURE [disk: DiskHandle] RETURNS [DeviceHandle] =
  BEGIN RETURN[LOOPHOLE[disk]] END;
```

```
  UnpackLabel: PROCEDURE [
    to: LONG POINTER TO PilotDisk.Label, from: LONG POINTER TO RDC.PackedLabel] =
    INLINE
    BEGIN
    to↑ ←
      [fileID:
       [SystemInternal.UniversalID[
         processor:
         [physical[
           a: from.processorID0, b: from.processorID1, c: from.processorID2]],
         sequence: LOOPHOLE[Environment.Long[
         num[
         lowbits: from.sequence1,
         highbits: from.sequence0Bit2*8192 + from.sequence0Bits3Thru15]]]],
        filePageLo: from.filePageLo, filePageHi: 0, immutable: from.immutable,
        temporary: from.temporary, zeroSize: from.zeroSize, type: from.type,
        bootChainLink: LOOPHOLE[LONG[0]]];
    IF from.type IN PilotFileTypes.PilotVFileType THEN
      to.filePageHi ← from.filePageHi
    ELSE to.bootChainLink ← LOOPHOLE[UnpackDiskAddress[from.bootChainLink]];
    END;

  END.
```

LOG

Time: August 8, 1979 1:09 AM By: Redell Action: Add log to file from Jarvis
Time: August 17, 1979 1:27 PM By: Gobbel Action: Pilot formatting
Time: August 22, 1979 12:40 PM By: Gobbel Action: Bring up to date with revis
***ed device face*
Time: October 2, 1979 11:38 PM By: Redell Action: Further cleanup, esp for bootchains
***(partial label verification)*
Time: October 8, 1979 9:22 AM By: McJones Action: Restart controller bug, add hardware
***State*
Time: October 10, 1979 3:53 PM By: McJones Action: INLINE procedures for sp
***eed; labelCheck is cue to call StartLabelFix*
Time: October 30, 1979 11:04 AM By: McJones AR2643: Label fix didn't reset ioc
***b.label from user's label*
Time: November 1, 1979 1:20 PM By: Frandeen Action: Fix recalibrate after heade
***r error; bug in SynchronizeWithController*
Time: November 7, 1979 1:17 PM By: McJones AR2739: Label fix up confused by
***spurious labelCheck status returned by read label command on sector with missing (label?) syn*
***ch byte*
Time: November 9, 1979 8:59 AM By: Frandeen Action: Implement runs of pages.
***Change interface to Poll. Change LabelFix procedures. Change interface to StartB since we no l*
***onger need an extra IOCB for label fixup.*
Time: December 13, 1979 11:29 AM By: Gobbel Action: Change name from SA400
***0D0Head to SA4000HeadD0*
Time: January 30, 1980 12:48 PM By: McJones Action: Update to match new face
***; add StartChain logic*
Time: June 25, 1980 10:34 AM By: McJones Action: 48-bit processor ids
Time: July 24, 1980 11:23 AM By: McJones Action: Label fixup must update iocb.operati
***on.clientHeader*

```
-- SA800HeadD0.mesa (last edited by: Forrest on: October 9, 1980 11:19 AM)

DIRECTORY
  DeviceCleanup USING [Await, Item, Reason],
  DOInputOutput USING [ControllerNumber, IOPage, GetNextController, fdc],
  HeadStartChain USING [Start],
  Inline USING [BITAND, BITOR, BITSHIFT],
  SA800Face USING [
    Attributes, Context, DeviceHandle, Function, OperationPtr, Status],
  SA800NeckD0,
  System USING [GetClockPulses, Pulses];

SA800HeadD0: PROGRAM
  IMPORTS
    DOInputOutput, DeviceCleanup, RemainingHeads: HeadStartChain, Inline,
    SA800NeckD0, System
  EXPORTS SA800Face, HeadStartChain =
  BEGIN OPEN SA800Face, SA800NeckD0;

  -- EXPORTED VARIABLES
  operationBlockLength: PUBLIC CARDINAL ← SIZE[IOCB] + 8;
  initialAllocationLength: PUBLIC CARDINAL ← operationBlockLength;
  nullDeviceHandle: PUBLIC DeviceHandle ← LOOPHOLE[-1];

  -- LOCAL CONSTANTS & VARIABLES
  task: UNSPECIFIED; --This variable has been shifted left to the proper postion.
  csb: CSBPtr ← NIL; --If this stays NIL, the controller did not respond
  loop: CARDINAL ← 0; --diagnostic counter (Re: QueueImmediate)--
  resetIOCB: IOCBlongPtr;
  formatTrackSize: CARDINAL = 80;
  -- value big enough for all format track operations
  state: State ←
    [user: [protect: FALSE, format: IBM, density: single, sectorLength: 256],
      head: [status: diskChange, sectorIndex: 2]];

  sectorMatrix: ARRAY SectorIndex OF SectorMatrix =
    [[sectorsPerTrack: 26, gap3: 27], -- 128 byte sectors (index = 0)
      [sectorsPerTrack: 15, gap3: 48], -- 256 byte sectors (index = 1)
      [sectorsPerTrack: 8, gap3: 90], -- 512 byte sectors (index = 2)
      [sectorsPerTrack: 4, gap3: 224]]; --1024 byte sectors (index = 3)

  wordsPerSector: ARRAY SectorIndex OF CARDINAL = [64, 128, 256, 512];

  commandMatrix: ARRAY Function OF CommandMatrix =
    [
      --nop--[nop, nopWakeAllow, NopStuff],
      --recalibrate--[seekTrack, seekWakeAllow, RecalibrateStuff],
      --recovery => noop--[nop, nopWakeAllow, NopStuff],
      --readSector--[readData, readWakeAllow, ReadSectorStuff],
      --writeSector--[writeData, writeWakeAllow, WriteSectorStuff],
      --writeDeletedSector--
      [writeDeletedData, writeWakeAllow, WriteDeletedSectorStuff],
      --readID--[readID, readWakeAllow, ReadIDStuff],
      --FormatTrack--[formatTrack, writeWakeAllow, FormatTrackStuff]];


  -- PROCS
```



```
  -- This procedure initiates an operation. The client passes an Operation block with most of the p
  **arameters included. We format the parameters that are unique to this implementation of the He
  **ad and microcode, add those not supplied by the client. The Operation block can then be calle
  **d an IOCB and may then be chained to the CSB. If the device is currently idle, set cmdWakeAllo
  **w and the device will generate a wakeup withing ~1.5 usecs. If it is not idle the microcode shoul
  **d get to the new IOCB if we just tack it to the end of the list.

  BuildStdFunction: PROC [op: OperationPtr, runContinuation: BOOLEAN ← FALSE]
    RETURNS [status: Status] =
    BEGIN OPEN Inline;
    iocb: IOCBlongPtr = LOOPHOLE[op + 8];

    -- If we have a diskChange status, then nothing else can be done.
    IF state.head.status.diskChange THEN {status ← diskChange; GOTO error};
    SELECT op.function FROM
      nop, readSector, readID, recalibrate, recovery => NULL;
      writeSector, writeDeletedSector, formatTrack =>
        IF state.user.protect OR state.head.status.writeProtect THEN {
          status ← writeProtect; GOTO error};
      ENDCASE;
    IF runContinuation THEN
      BEGIN -- don't clobber next, softwareNext
      softwareNextCopy: IOCBlongPtr = iocb.softwareNext;
      nextCopy: IOCBlongPtr = iocb.next;
      Clear16Words[iocb];
      iocb.softwareNext ← softwareNextCopy;
      iocb.next ← nextCopy;
      iocb.wake ← commandMatrix[op.function].wake;
      iocb.result ← clearResult;
      iocb.function ← commandMatrix[op.function].function;
      commandMatrix[op.function].stuffer[iocb, op];
      RunThisIOCB[iocb] -- queues already OK

      END
    ELSE
      BEGIN
      Clear16Words[iocb];
      iocb.wake ← commandMatrix[op.function].wake;
      iocb.function ← commandMatrix[op.function].function;
      iocb.result ← clearResult;
      commandMatrix[op.function].stuffer[iocb, op];
      IF op.count # 0 THEN OnIOCBchain[iocb]; -- ELSE nothingToDo

      END;
    status ← BITAND[state.head.status, infoMask];
    status.inProgress ← op.count # 0;
    status.busy ← csb.state.next # NIL;
    EXITS error => status ← BITOR[status, BITAND[state.head.status, infoMask]];
    END;

  Clear16Words: PROC [pointer: LONG POINTER] = INLINE {
    LOOPHOLE[pointer, LONG POINTER TO ARRAY [0..16) OF WORD]↑ ← ALL[0]};

  DiskChangeClear: PUBLIC PROC [device: DeviceHandle] = {
    state.head.status.diskChange ← FALSE; Reset[device]};

  GetContext: PUBLIC PROC [DeviceHandle] RETURNS [Context] = {RETURN[state.user]};
```

```
GetDeviceAttributes: PUBLIC PROC [DeviceHandle] RETURNS [Attributes] =
  BEGIN
  RETURN[
    [deviceType: SA800, numberOfCylinders: 77, numberOfHeads: 1,
      trackLength: formatTrackSize]]
  END;

GetNextDevice: PUBLIC PROC [device: DeviceHandle] RETURNS [DeviceHandle] =
  BEGIN OPEN D0InputOutput, Inline;
  RETURN[
    SELECT TRUE FROM
      device = nullDeviceHandle =>
        IF csb = NIL THEN nullDeviceHandle
        ELSE BITSHIFT[GetNextController[fdc, 0], 8],
      -- IF controller ← GetNextController[fdc, disk.controller] # 0 THEN
      --  the world has decided on multiple drives/controller.
      --  This code is not intended to handle that situation.

      ENDCASE => nullDeviceHandle];
  END;

-- This code will only handles a single controller/single device configuration.
-- It does check for no controller.

Initialize: PUBLIC PROC [notify: WORD, initialAllocation: LONG POINTER] =
  BEGIN
  IF csb # NIL THEN
    BEGIN
    csb.state ←
      [next: NIL, abortMicrocode: 0, notify: Inline.BITOR[notify, 100000B],
        tail: NIL];
    -- The resetIOCB is used by SetChipParms.  It acutally uses it three (3)
    -- times during the SetChipParms, but that is of little consequence here.
    resetIOCB ← initialAllocation + 8;
    Reset[nullDeviceHandle]; --Pass initial parameters to FDC chip

    END;
  END;

--This routine will wait for the IOCB chain to empty before letting the debugger get control.  For t
**he time being (i.e., during debugging) it will also release after a ~15 second wait.  The theory is t
**hat if the queue did not go empty in 15 seconds, it never will, and the debugger is going to aid in
**finding out why the hell we got here to start with.

InitializeCleanup: PUBLIC PROC [device: DeviceHandle] =
  BEGIN
  wait: System.Pulses = [15000000/LONG[64]];
  item: DeviceCleanup.Item;
  reason: DeviceCleanup.Reason;
  csbState: CSBState;
  DO
    reason ← DeviceCleanup.Await[@item];
    SELECT reason FROM
      turnOff, kill =>
        BEGIN
        clock: System.Pulses = System.GetClockPulses[];
        WHILE csb.state.next # NIL AND (System.GetClockPulses[] - clock) <= wait
          DO ENDLOOP;
        csbState ← csb.state;
```



```
        END;
      turnOn => csb.state ← csbState;
      ENDCASE;
    ENDLOOP;
  END;

Initiate: PUBLIC PROC [operationPtr: OperationPtr] RETURNS [Status, CARDINAL] =
  BEGIN
  SELECT operationPtr.function FROM
    nop, recalibrate, recovery => operationPtr.count ← 1;
    ENDCASE;
  RETURN[
    BuildStdFunction[operationPtr],
      --800ms max seek time + tracks*(3ms step + 125ms/rotation)
      3 + (operationPtr.count/20)]
  END;

--The iocb is ready to go on the CSB chain.  If csb.state.next = state.head.tail = NIL then create
**an initial entry and start the microcode by setting cmdWakeAllow.  The microcode should wake
**within 15 usecs.
-- There is a race condition regarding an IOCB completing while we are chaining.
-- This is is handled in Poll (and in reset)...

OnIOCBchain: PROC [iocb: IOCBlongPtr] =
  BEGIN
  iocb.next ← iocb.softwareNext ← NIL;
  IF csb.state.next = NIL THEN {csb.state.tail ← iocb; RunThisIOCB[iocb]}
  ELSE
    BEGIN
    IF LOOPHOLE[csb.state.tail - 8, OperationPtr].count <= 1 THEN
      csb.state.tail.next ← iocb;
    -- place on hardware queue if tail op will complete
    csb.state.tail.softwareNext ← iocb; -- always place on software queue
    csb.state.tail ← iocb; -- update tail to point to new IOCB

    END;
  END;

Poll: PUBLIC PROC [operationPtr: OperationPtr] RETURNS [status: Status] =
  BEGIN
  iocb: IOCBlongPtr = LOOPHOLE[operationPtr + 8];
  status ← ResultToStatus[iocb];
  IF status.inProgress THEN RETURN
  ELSE
    IF status.error THEN
      BEGIN
      IF iocb.next = NIL AND iocb.softwareNext # NIL THEN
        RunThisIOCB[iocb.softwareNext];
      RETURN; -- an error occured in the middle of a run. Start the next IOCB

      END
    ELSE -- status = goodCompletion
      BEGIN -- ASSUMES 1 HEAD, SINGLE DENSITY IBM FORMAT
      SELECT operationPtr.function FROM
        nop, recalibrate, recovery => NULL;
        readSector, writeDeletedSector, writeSector =>
          BEGIN OPEN da: operationPtr.address;
          IF (da.sector ← da.sector + 1) > sectorMatrix[
            state.head.sectorIndex].sectorsPerTrack THEN {
```

```
            da.sector ← 1; da.cylinder ← da.cylinder + 1};
          IF operationPtr.incrementDataPointer THEN
            BEGIN OPEN b: operationPtr.buffer;
            b.address ← b.address + wordsPerSector[state.head.sectorIndex];
            b.length ← b.length - wordsPerSector[state.head.sectorIndex]
            END;
          END;
        readID =>
          BEGIN OPEN da: operationPtr.address;
          IF (da.sector ← da.sector + 1) > sectorMatrix[
            state.head.sectorIndex].sectorsPerTrack THEN {
            da.sector ← 1; da.cylinder ← da.cylinder + 1};
          IF operationPtr.incrementDataPointer THEN
            BEGIN OPEN b: operationPtr.buffer;
            b.address ← b.address + 3;
            b.length ← b.length - 3
            END;
          END;
        formatTrack =>
          BEGIN
          operationPtr.address.cylinder ← operationPtr.address.cylinder + 1;
          operationPtr.address.sector ← 1;
          IF operationPtr.incrementDataPointer THEN
            BEGIN OPEN b: operationPtr.buffer;
            b.address ← b.address + formatTrackSize;
            b.length ← b.length - formatTrackSize
            END;
          END;
        ENDCASE;
      -- get things moving
      IF (operationPtr.count ← operationPtr.count - 1) = 1 THEN
        iocb.next ← iocb.softwareNext; -- will be last page of run
      IF operationPtr.count # 0 THEN
        BEGIN
        status ← BuildStdFunction[operationPtr, TRUE];
        IF ~status.inProgress AND iocb.next # NIL THEN RunThisIOCB[iocb.next];
        END;
      END;
    -- we are here if we are done with this IOCB, and we didn't explicitly start another.
    -- check for the race mentioned in OnIOCBChain
    IF iocb.next # NIL AND csb.state.next = NIL AND ResultToStatus[
      iocb.next].inProgress THEN RunThisIOCB[iocb.next];
    END;

  ResultToStatus: PROC [iocb: IOCBlongPtr] RETURNS [status: Status] =
    BEGIN
    lateDMA: Completion = [type: 1, code: 1];
    sectorNotFound: Completion = [type: 3, code: 0];
    idCRCerror: Completion = [type: 1, code: 2];
    dataCRCerror: Completion = [type: 1, code: 3];
    notReady: Completion = [type: 2, code: 0];
    track00NotFound: Completion = [type: 2, code: 2];
    clearStdResult: StandardResult =
      [deleted: FALSE, completion: [type: 0B, code: 0B]];
    controllerError: ControllerStatus =
      [dmaWakeReq: TRUE, oDataParity: TRUE, oFault: TRUE, fdcIntF: FALSE];
    result: Result ← iocb.result;
    IF result # clearResult THEN
```



```
      BEGIN
      status ←
        [diskChange: state.head.status.diskChange
        -- OR (~state.head.status.notReady # result.driveStatus.ready0),
        OR (state.head.status.notReady = result.driveStatus.ready0), na1: FALSE,
        twoSided: FALSE, na3: FALSE,
        error: (result.standardResult.completion.type # 0) OR
        (Inline.BITAND[result.controllerStatus, controllerError] # 0),
        inProgress: FALSE,
        recalibrateError: result.standardResult.completion = track00NotFound,
        dataLost: result.standardResult.completion = lateDMA OR iocb.length # 0,
        notReady: result.standardResult.completion = notReady OR
        ~result.driveStatus.ready0,
        writeProtect: result.driveStatus.writeProtect OR state.user.protect,
        deletedData: result.standardResult.deleted,
        recordNotFound: result.standardResult.completion = sectorNotFound,
        crcError: result.standardResult.completion = idCRCerror OR
        result.standardResult.completion = dataCRCerror,
        track00: result.driveStatus.track00,
        index: result.driveStatus.indexPulse, busy: FALSE];
      status.error ← (Inline.BITAND[status, statusError] # 0) OR
        status.diskChange;
      state.head.status ← status; --save most recent status for drive
      -- status.diskChange ← FALSE; + + only returned on Initiate

      END
    ELSE {status ← inProgress; status.busy ← csb.state.next # NIL};
    END;

  Reset: PUBLIC PROC [device: DeviceHandle] =
    BEGIN OPEN SA800NeckD0;
    op: OperationPtr = LOOPHOLE[resetIOCB - 8];
    revolutionTime: System.Pulses = [LONG[2000]]; --125 msecs +/-
    reset: UNSPECIFIED = 120001B; --Command to reset chip
    rp0: UNSPECIFIED = 120000B; --Parameter 0 for chip reset
    current: IOCBlongPtr;
    -- Constants unique to setting initial chip parameters
    initialize: CARDINAL = 15B; --specify initialize subcommand
    surface0: CARDINAL = 20B; --specify surface0
    surface1: CARDINAL = 30B; --specify surface1
    stepRate: CARDINAL = 10; --millisecond stepping rate
    headSettle: CARDINAL = 15;
    --millisecond time to let head settle after stepping
    headUnload: CARDINAL = 4; --revolutions before head is unloaded
    headLoad: CARDINAL = 36; --milliseconds allowed to load heads (MOD4)
    specifyInit: IOCB =
      [next: NIL, softwareNext: NIL, address: NIL, length: 0, result: clearResult,
        wake: initWakeAllow, function: specify,
        p0: [parameter, parm1, initialize], p1: [parameter, parm2, stepRate],
        p2: [parameter, parm3, headSettle],
        p3: [parameter, 0, headUnload*16 + headLoad/4], p4: LOOPHOLE[0],
        unused: 0];
    specifybadTracks: IOCB =
      [next: NIL, softwareNext: NIL, address: NIL, length: 0, result: clearResult,
        wake: initWakeAllow, function: specify, p0: [parameter, parm1, surface0],
        p1: [parameter, parm2, 377B], p2: [parameter, parm3, 377B],
        p3: [parameter, 0, 377B], p4: LOOPHOLE[0], unused: 0];
    dataWindow: CARDINAL ← 344B; --data window value for 250Kbit/sec rate
```

```
-- This is going to put the IOCB on a presumed empty chain and WAIT FOR IT TO
-- COMPLETE! If it doesn't complete in "n" iterations through the loop,
-- well....sheeeeeeit! NOTE: The counter "loop" is the maximum number of times
-- we had to stay in this loop waiting for the microcode to finish an operation.
-- If it every hits 'n', the magic number, we have failed. Since this is pure
-- microcode and no data transfer, that's not too bad. For you floppy disk wizards,
-- the variable loop in the global frame is the maximum number of times we stayed in
-- this loop. Should this operation be in doubt, look there for hints. If one
-- feels confident, take out the maintenance of the counter.
QueueImmediate: PROC [iocb: IOCBlongPtr] =
  BEGIN
  index: CARDINAL;
  iocb.next ← NIL;
  RunThisIOCB[iocb];
  FOR index IN [0..1000) DO IF csb.next = NIL THEN EXIT; loop ← loop ENDLOOP;
  IF index > loop THEN loop ← index;
  END;

--Reset the controller so we don't get any more wakes.
OUTPUT[0, task + 0]; --write zero to reset register
OUTPUT[0, task + 1B]; --write zero to wake allow register
OUTPUT[1B, task + 0]; --request controller master reset
IF (current ← csb.state.next) # NIL THEN
  BEGIN
  --The microcode was "busy" by the definition that the IOCB chain is not empty.
  -- Since the client is calling us, this operation must be taking longer than
  -- it should, whatever that is. We want to kill the current operation, forcing
  -- the microcode to proceed down the merry path of IOCBs. First clear the current
  -- operation from the chip be issuing a reset to the chip.
  clock: System.Pulses = System.GetClockPulses[];
  -- Reset the FDC chip.
  -- (This probably loses all initialization parameters. We will assume it does.)
  OUTPUT[reset, task + 6];
  OUTPUT[rp0, task + 6];
  --In order for Reset to work, the chain must be empty. To do that, we need to
  -- save the address of the next (logically) IOCB to be activated after the one
  -- we blow away. Then we can restore it in the CSB when we finish.
  current.next ← NIL;
  -- ask the read loop to stop
  csb.state.abortMicrocode ← -1;
  --Start the microcode again with an index wake
  -- (which includes IOATTEN so DMA requests will complete).
  UNTIL csb.state.next = NIL DO
    OUTPUT[3B, task + 1B];
    --Wait for the microcode to finish off the current operation. This could take
    -- a full revolution of the diskette!!!! Note that this wake generation is
    -- coming from the controller, not the FDC chip. If we don't get the wake,
    -- ever, then we will hang inside the outer loop, FOREVER (give or take
    -- a microsecond).
    WHILE (clock - System.GetClockPulses[]) < revolutionTime DO
      --ARGH!!!
      IF csb.state.next = NIL THEN EXIT; --inner loop only

      ENDLOOP;
    ENDLOOP;
  csb.state.abortMicrocode ← 0;
  END;
-- Well! We just wiped out all the parameters set up during initialization.
```



```
  -- Guess we better set them back before anybody notices.
  OUTPUT[dataWindow, task + 4B];
  resetIOCB↑ ← specifyInit;
  QueueImmediate[resetIOCB];
  resetIOCB↑ ← specifybadTracks;
  QueueImmediate[resetIOCB];
  resetIOCB.p0 ← [parameter, parm1, surface1];
  resetIOCB.result ← clearResult;
  QueueImmediate[resetIOCB];
  [] ← ResultToStatus[resetIOCB];
  -- Request track00 since the chip no longer knows where it is. If this fails,
  -- no one will ever know. If this fails, no one will ever know, but chances of
  -- future success are dim.
  op.function ← recalibrate;
  [] ← Initiate[op];
  END;

RunThisIOCB: PROC [iocb: IOCBlongPtr] = INLINE {
  csb.state.next ← iocb; OUTPUT[5B, task + 1B]};

SetContext: PUBLIC PROC [device: DeviceHandle, context: Context]
  RETURNS [ok: BOOLEAN] =
  BEGIN
  IF
    (ok ← (context.format # Troy) AND (context.density # double) AND
      (SELECT context.sectorLength FROM
        64, 128, 256, 512 => TRUE,
        ENDCASE => FALSE)) THEN state.user ← context;
  state.head.sectorIndex ←
    SELECT state.user.sectorLength FROM
      64 => 0,
      128 => 1,
      256 => 2,
      ENDCASE => 3;
  END;

Start: PUBLIC PROC = {RemainingHeads.Start[]}; -- exported to StartChain

StartHead: PROC =
  BEGIN OPEN DOInputOutput, Inline;
  controller: ControllerNumber = GetNextController[fdc, 0];
  IF controller # 0 THEN {
    csb ← LOOPHOLE[@IOPage[controller]]; task ← BITSHIFT[controller, 4]}
  END;

-- Procedures called by BuildStdFunction via Commandmatrix to stuff IOCB's

FormatTrackStuff: PROC [iocb: IOCBlongPtr, operation: OperationPtr] =
  BEGIN
  RecordId: TYPE = MACHINE DEPENDENT RECORD [
    cyl: [0..256), head: [0..256), sector: [0..256), sectorLength: [0..256)];
  gap1: CARDINAL = 26;
  gap5: CARDINAL = 40;
  gap3: CARDINAL = sectorMatrix[state.head.sectorIndex].gap3;
  numberOfSectors: CARDINAL = sectorMatrix[
    state.head.sectorIndex].sectorsPerTrack;
  p1: ChipLoad = [parameter, parm2, gap3]; --intersector gap--
  p2: ChipLoad = --sectors/track & length--
```

```
    [parameter, parm3,
      numberOfSectors + Inline.BITSHIFT[state.head.sectorIndex, 5]];
  p3: ChipLoad = [parameter, parm4, gap5]; --last sector to index address mark--
  p4: ChipLoad = [parameter, 0, gap1]; --index to first ID record address mark--
  trackBuffer: LONG POINTER TO RecordId ← operation.buffer.address;
  FOR index: CARDINAL IN (0..numberOfSectors] DO
    trackBuffer↑ ←
      [cyl: operation.address.cylinder, head: 0, sector: index,
        sectorLength: state.head.sectorIndex];
    trackBuffer ← trackBuffer + SIZE[RecordId];
    ENDLOOP;
  iocb.address ← operation.buffer.address;
  iocb.length ← SIZE[RecordId]*numberOfSectors;
  iocb.p0 ← [parameter, --track address --parm1, operation.address.cylinder];
  iocb.p1 ← p1;
  iocb.p2 ← p2;
  iocb.p3 ← p3;
  iocb.p4 ← p4;
  END;

NopStuff: PROC [iocb: IOCBlongPtr, operation: OperationPtr] = {};

ReadIDStuff: PROC [iocb: IOCBlongPtr, operation: OperationPtr] =
  BEGIN
  iocb.address ← operation.buffer.address;
  iocb.length ← MIN[2, operation.buffer.length];
  iocb.p0 ← [parameter, parm1, operation.address.cylinder];
  iocb.p1 ← [parameter, parm2, 0];
  iocb.p2 ← [parameter, 0, 1B];
  END;

ReadSectorStuff, WriteSectorStuff, WriteDeletedSectorStuff: PROC [
  iocb: IOCBlongPtr, operation: OperationPtr] =
  BEGIN
  iocb.address ← operation.buffer.address;
  iocb.length ← MIN[
    operation.buffer.length, wordsPerSector[state.head.sectorIndex]];
  iocb.p0 ← [parameter, parm1, operation.address.cylinder];
  iocb.p1 ← [parameter, parm2, operation.address.sector];
  iocb.p2 ← [parameter, 0, Inline.BITSHIFT[state.head.sectorIndex, 5] + 1B];
  END;

RecalibrateStuff: PUBLIC PROC [iocb: IOCBlongPtr, operation: OperationPtr] = {
  iocb.p0 ← [parameter, 0, 0] --track ← zero--};

--Mainline Code

StartHead[];
END.
```

LOG

Edited: June 17, 1980 9:53 AM by AOF: Added log to file.
Edited: June 18, 1980 9:09 AM by AOF: Added additional enumerated element (recovery) to functi
***ons allowed in Initiate.*
Edited: June 27, 1980 10:59 AM by AOF: Changed definition of Reset and restructured code som
***ewhat.*
Edited: September 15, 1980 3:59 PM by Forrest: Runs of pages (software).
Edited: September 17, 1980 3:13 PM by Forrest: Redefine Disk Change.
Edited: October 2, 1980 1:31 PM by Luniewski: status.diskChange => status.error.

Edited: October 9, 1980 11:18 AM by Forrest: Debug Runs of Pages.

```
-- SetGMTUsingEthernet (edited by: BLyon on: August 20, 1980 2:19 PM)

DIRECTORY
  EthernetOneFace USING [
    AddCleanup, ControlBlock, DeviceHandle, GetEthernet1Address, GetNextDevice,
    GetPacketLength, GetStatus, GlobalStatePtr, globalStateSize, nullDeviceHandle,
    QueueInput, QueueOutput, RemoveCleanup, TurnOff, TurnOn],
  Environment USING [Long],
  Inline USING [BITAND],
  ProcessorFace USING [SetGreenwichMeanTime],
  TemporarySetGMT USING [];

SetGMTUsingEthernet: PROGRAM
  IMPORTS EthernetOneFace, Inline, ProcessorFace EXPORTS TemporarySetGMT =
  BEGIN
  SetGMT: PUBLIC PROCEDURE =
    BEGIN
    IF EthernetExists[] THEN ProcessorFace.SetGreenwichMeanTime[GetTime[].time];
    END;

  Byte: TYPE = [0..377B];

  -- Following started as a copy of GTime.mesa as edited by Johnsson on July 12, 1979 8:31 AM
  NetTime: TYPE = MACHINE DEPENDENT RECORD [
    -- this is what comes in an AltoTimeReply Pup
    timeHigh, timeLow: CARDINAL,
    direction: WestEast,
    zone: [0..127],
    zoneMinutes: [0..255],
    beginDST, endDST: CARDINAL];

  WestEast: TYPE = {West, East};

  TimeData: TYPE = RECORD [
    direction: WestEast,
    zone: [0..12],
    beginDST: [0..366],
    zoneMinutes: [0..59],
    endDST: [0..366]];

  PacketTypePup: CARDINAL = 1000B;
  AltoTimeRequest: CARDINAL = 206B;
  AltoTimeReply: CARDINAL = 207B;
  SocMiscServices: CARDINAL = 4;

  PupHeader: TYPE = MACHINE DEPENDENT RECORD [
    eDest, eSource: Byte,
    eWord2, pupLength: CARDINAL,
    transportControl, pupType: Byte,
    pupID1, pupID2: CARDINAL,
    destNet, destHost: Byte,
    destSocket1, destSocket2: CARDINAL,
    sourceNet, sourceHost: Byte,
    sourceSocket1, sourceSocket2: CARDINAL,
    xSum: CARDINAL];

  PupOverhead: CARDINAL = SIZE[PupHeader];
```



```
  Pup: TYPE = MACHINE DEPENDENT RECORD [
    head: PupHeader, data: ARRAY [0..0) OF WORD];

  DriverNeedsSomeGlobalStorage: PUBLIC ERROR = CODE;


  GetTime: PROC RETURNS [time: LONG CARDINAL, timeData: TimeData] =
    BEGIN OPEN EthernetOneFace;
    ether: DeviceHandle = GetNextDevice[nullDeviceHandle];
    global: GlobalStatePtr; -- Allocate space when needed
    -- This works only if our MDS is in the first 64K
    iocblock: ARRAY [0..20 --controlBlockSize-- + 3) OF CARDINAL;
    iocb: ControlBlock = LOOPHOLE[Inline.BITAND[@iocblock + 3, -4], POINTER];
    inputSize: CARDINAL = PupOverhead + SIZE[NetTime] + slop;
    slop: CARDINAL = 12;
    ipup: ARRAY [0..inputSize + 3) OF CARDINAL;
    opup: ARRAY [0..PupOverhead + 3) OF CARDINAL;
    request: POINTER TO PupHeader = Inline.BITAND[@opup + 3, -4];
    answer: POINTER TO Pup = Inline.BITAND[@ipup + 3, -4];
    myHost: Byte;
    id1: WORD = 1234B;
    id2: WORD = 56710B;


    IF globalStateSize # 0 THEN ERROR DriverNeedsSomeGlobalStorage;


      [, myHost] ← GetEthernet1Address[ether];
    request↑ ←
      [eDest: 0, eSource: myHost, eWord2: PacketTypePup,
        pupLength: 2*(PupOverhead - 2), transportControl: 0,
        pupType: AltoTimeRequest, pupID1: id1, pupID2: id2, destNet: 0,
        destHost: 0, destSocket1: 0, destSocket2: SocMiscServices, sourceNet: 0,
        sourceHost: myHost, sourceSocket1: 0, sourceSocket2: SocMiscServices,
        xSum: 177777B];

    AddCleanup[ether];
    THROUGH [0..3) DO
      THROUGH [0..3) DO
        TurnOff[ether];
        TurnOn[ether, myHost, 0, 0, global];
        answer.head.eWord2 ← 0;
        QueueOutput[ether, request, PupOverhead, iocb];
        THROUGH [0..LAST[CARDINAL]/2) DO
          IF GetStatus[iocb] # pending THEN
            BEGIN
            IF GetStatus[iocb] = ok AND GetPacketLength[iocb] =
              PupOverhead + SIZE[NetTime] AND answer.head.eWord2 = PacketTypePup
              AND answer.head.pupType = AltoTimeReply AND answer.head.pupID1 = id1
              AND answer.head.pupID2 = id2 AND answer.head.destHost = myHost AND
              answer.head.destSocket1 = 0 AND answer.head.destSocket2 =
              SocMiscServices AND answer.head.sourceSocket1 = 0 AND
              answer.head.sourceSocket2 = SocMiscServices THEN
              BEGIN
              nt: POINTER TO NetTime = LOOPHOLE[@answer.head.xSum];
              LOOPHOLE[time, Environment.Long] ←
                [num[lowbits: nt.timeLow, highbits: nt.timeHigh]];
              timeData ←
```

```
            [direction: nt.direction, zone: nt.zone, beginDST: nt.beginDST,
              zoneMinutes: nt.zoneMinutes, endDST: nt.endDST];
            TurnOff[ether];
            RemoveCleanup[ether];
            RETURN
            END;
          -- Start (another) read either because the send just finshed or we didn't like the packet t
**hat just arrived.
          QueueInput[ether, answer, inputSize, iocb];
          END;
        ENDLOOP;
      ENDLOOP;
    ENDLOOP;
  TurnOff[ether];
  RemoveCleanup[ether];
  time ← 0;
  RETURN
  END;

EthernetExists: PROCEDURE RETURNS [BOOLEAN] =
  BEGIN OPEN EthernetOneFace;
  RETURN[GetNextDevice[nullDeviceHandle] # nullDeviceHandle];
  END;

END.
LOG
Time: April 15, 1980 1:23 PM  By: McJones  Action: Create from SystemImpl of February
**26, 1980 12:22 PM
Time: April 21, 1980 2:16 PM  By: Murray  Action: Facification, trap for globalState
Time: June 26, 1980 5:02 PM  By: McJones  Action: OISProcessorFace = >ProcessorFac
**e
Time: July 30, 1980 1:42 PM  By: Murray/Howard  Action: Fix bugs if timeouts occur
Time: August 20, 1980 2:18 PM By: BLyon Action: renamed EthernetFace to EthernetOneFace
```

```
-- UserTerminalHeadD0.mesa (last edited by: McJones on: July 31, 1980 9:02 AM)

DIRECTORY
  BitBlt USING [BBTable],
  D0InputOutput USING [
    ControllerNumber, ControllerType, GetNextController, Input, null,
    nullControllerNumber, uibScb, utvfc],
  DeviceCleanup USING [Await, Item, Reason],
  DisplayFace USING [Background, Cursor, CursorPtr, GlobalStatePtr, Point],
  Environment USING [bitsPerWord, first64K, PageCount, PageNumber, wordsPerPage],
  HeadStartChain USING [Start],
  Inline USING [BITOR, LongMult, LowHalf],
  KeyboardFace USING [],
  KeyStations USING [KeyBits],
  MouseFace USING [Buttons, Point],
  ProcessorFace USING [GetClockPulses, microsecondsPerHundredPulses],
  RuntimeInternal USING [WorryCallDebugger];

UserTerminalHeadD0: PROGRAM
  IMPORTS
    D0InputOutput, DeviceCleanup, RemainingHeads: HeadStartChain, Inline,
    ProcessorFace, RuntimeInternal
  EXPORTS DisplayFace, HeadStartChain, KeyboardFace, MouseFace, ProcessorFace =
  BEGIN
  -- Types
  CSBPtr: TYPE = LONG POINTER TO CSBState;
  CSBState: TYPE = MACHINE DEPENDENT RECORD [
    dcbChainHead: PDCB, wakeupMask: WORD];
  PDCB: TYPE = POINTER TO DCB;
  DCB: TYPE = MACHINE DEPENDENT RECORD [
    -- address must be even
    next: PDCB,
    resolution: {high, low},
    background: DisplayFace.Background,
    indenting: [0..77B], -- in units of 16 bits
    width: [0..377B], -- in units of 16 bits; must be even
    shortBitmap: POINTER, -- must be even
    tag: {short, long},
    height: [0..77777B], -- in double scan lines
    longBitmap: LONG POINTER]; -- must be even
  UTVFCControlRegister: TYPE = MACHINE DEPENDENT RECORD [
    -- not currently used
    controlNibbles: [0..377B] ← 0,
    IncNC: [0..1] ← 0,
    PPBckGnd: [0..3] ← 0,
    PPBlank, PPVS, PreOField, AllowWU, ClrNC: [0..1] ← 0];
  UTVFCInputRegister: TYPE = MACHINE DEPENDENT RECORD [
    controllerIDa: [0..377B] ← 2,
    bitClockRate: [0..37B],
    controllerIDb: [0..3] ← 2,
    test: [0..1]];
  cslBitClockRate: [0..37B] = 5B;
  lfBitClockRate: [0..37B] = 3B;
  -- Hardware constants
  csbPtr: CSBPtr = LOOPHOLE[LONG[420B]];
  pagesForBitmap: PUBLIC Environment.PageCount; -- see initialization below
  -- Size of screen image:
  height: PUBLIC CARDINAL [0..32767] ← 808; -- LF and CSL
```



```
  torHeight: CARDINAL = 800;
  width: PUBLIC CARDINAL [0..32767] ← lfWordsPerLine*Environment.bitsPerWord;
  -- see initialization below
  cslWordsPerLine: CARDINAL = 38;
  lfWordsPerLine: CARDINAL = 64;
  torWordsPerLine: CARDINAL = 40;
  -- Size of bitmap (possibly large than screen image):
  bitmapWidth: CARDINAL ← lfWordsPerLine*Environment.bitsPerWord;
  -- see initialization below
  pixelsPerInch: PUBLIC CARDINAL ← 72;
  hasBuffer: PUBLIC BOOLEAN ← FALSE;
  refreshRate: PUBLIC CARDINAL ← lfRefreshRate; -- frames (two fields) per second
  cslRefreshRate: CARDINAL = 30;
  lfRefreshRate: CARDINAL = 38; -- actually ??
  interlaced: PUBLIC BOOLEAN ← TRUE; -- Is the refresh two-way interlaced?
  hasBorder: PUBLIC BOOLEAN ← FALSE; -- is SetBorderPattern legal?
  keyboard: PUBLIC LONG POINTER TO READONLY KeyStations.KeyBits ← LOOPHOLE[LONG[
    177033B]];
  millisecondsPerTick: PUBLIC CARDINAL ← lfMillisecondsPerTick;
  -- exported to ProcessorFace
  cslMillisecondsPerTick: CARDINAL = 50;
  lfMillisecondsPerTick: CARDINAL = 40; -- actually 39.7
  torMillisecondsPerTick: CARDINAL = 40; -- what should this be?
  cursorPosition: PUBLIC LONG POINTER TO DisplayFace.Point ← LOOPHOLE[LONG[426B]];
  pHardwareCursor: DisplayFace.CursorPtr = LOOPHOLE[LONG[431B]];
  pDCBNull: PDCB = LOOPHOLE[0];
  microsecondsPerScanLine: CARDINAL ← 14;
  -- Global variables
  displayState: {disconnected, off, on} ← disconnected;
  bitmapBase: LONG POINTER; -- undefined if displayState = disconnected
  controllerType: D0InputOutput.ControllerType;
  controller: D0InputOutput.ControllerNumber;
  displayOn: BOOLEAN ← FALSE;
  displayConnected: BOOLEAN ← FALSE;
  globalStateSize: PUBLIC CARDINAL ← SIZE[DCB]*2;
  pDCBReal: LONG POINTER TO DCB;
  pDCBBlank: LONG POINTER TO DCB;
  ErrorHalt: PROCEDURE =
    BEGIN RuntimeInternal.WorryCallDebugger["Error in DisplayHeadD0"L]; END;
  -- Mouse

  buttons: PUBLIC LONG POINTER TO READONLY MouseFace.Buttons ← LOOPHOLE[keyboard];
  mouse: LONG POINTER TO MouseFace.Point = LOOPHOLE[LONG[424B]];
  position: PUBLIC LONG POINTER TO READONLY MouseFace.Point ← mouse;
  SetPosition: PUBLIC PROCEDURE [newMousePosition: MouseFace.Point] =
    BEGIN mouse↑ ← newMousePosition END;
  -- Cursor

  SetCursorPattern: PUBLIC PROCEDURE [cursorPtr: DisplayFace.CursorPtr] =
    BEGIN pHardwareCursor↑ ← cursorPtr↑ END;
  -- Bitmaps

  Connect: PUBLIC PROCEDURE [bitmap: Environment.PageNumber] =
    -- Connects display to specified bitmap and leaves it in off state.
    BEGIN
    displayState ← off;
    bitmapBase ← LOOPHOLE[Inline.LongMult[bitmap, Environment.wordsPerPage]];
    -- LongPointerFromPage
```

```
    csbPtr.dcbChainHead ← Inline.LowHalf[pDCBBlank]; -- DCB's are in first64K

    END;

  Disconnect: PUBLIC PROCEDURE =
    -- Places display in disconnected state, turning screen black, and minimizing consumption of r
**esources.
    BEGIN displayState ← disconnected; csbPtr.dcbChainHead ← pDCBNull; END;

  TurnOn: PUBLIC PROCEDURE =
    -- Display must have been connected; places display in on state, causing bitmap to be displaye
**d on screen.
    BEGIN
    IF displayState = disconnected THEN ErrorHalt[];
    displayState ← on;
    pDCBReal.longBitmap ← bitmapBase;
    csbPtr.dcbChainHead ← Inline.LowHalf[pDCBReal] -- DCB's are in first64K

    END;

  TurnOff: PUBLIC PROCEDURE =
    -- Display must have been connected; places display in off state, causing background color to
**be displayed on screen.
    BEGIN
    IF displayState = disconnected THEN ErrorHalt[];
    displayState ← off;
    csbPtr.dcbChainHead ← Inline.LowHalf[pDCBBlank]; -- DCB's are in first64K

    END;

  GetBitBltTable: PUBLIC PROCEDURE RETURNS [BitBlt.BBTable] =
    BEGIN
    IF displayState = disconnected THEN ErrorHalt[];
    RETURN[
      [dst: [word: bitmapBase, bit: 0], dstBpl: bitmapWidth,
        src: [word: bitmapBase, bit: 0], srcDesc: [srcBpl[bitmapWidth]],
        width: width, height: height, flags: []]];
    END;

  SetBorderPattern: PUBLIC PROCEDURE [oddPairs, evenPairs: [0..377B]] = BEGIN END;

  SetBackground: PUBLIC PROCEDURE [background: DisplayFace.Background] =
    BEGIN pDCBReal.background ← pDCBBlank.background ← background END;

  InitializeCleanup: PUBLIC PROCEDURE =
    BEGIN OPEN DeviceCleanup;
    item: Item;
    reason: Reason;
    state: CSBState;
    mouseCoord: MouseFace.Point;
    cursorCoord: DisplayFace.Point;
    cursor: DisplayFace.Cursor;
    timeDone: LONG CARDINAL;
    maxPulsesPerRefresh: LONG CARDINAL =
      2* --for safety--
        (LONG[100]* --pulsesPerHundredPulses--1000000 --microsecondsPerSecond--
          )/(LONG[refreshRate] --framesPerSecond--
              *ProcessorFace.microsecondsPerHundredPulses);
```



```
    DO
      reason ← Await[@item];
      SELECT reason FROM
        turnOff =>
          BEGIN
          state ← csbPtr↑;
          mouseCoord ← mouse↑;
          cursorCoord ← cursorPosition↑;
          cursor ← pHardwareCursor↑;
          csbPtr.dcbChainHead ← pDCBNull;
          timeDone ← ProcessorFace.GetClockPulses[];
          WHILE ProcessorFace.GetClockPulses[] - timeDone < maxPulsesPerRefresh DO
            ENDLOOP;
          END;
        turnOn =>
          BEGIN
          mouse↑ ← mouseCoord;
          cursorPosition↑ ← cursorCoord;
          pHardwareCursor↑ ← cursor;
          csbPtr↑ ← state;
          END;
        ENDCASE;
      ENDLOOP;
    END;

  Initialize: PUBLIC PROCEDURE [
    globalState: DisplayFace.GlobalStatePtr, wakeVF: WORD] =
    BEGIN
    dcb: DCB =
      [next: pDCBNull, resolution: high, background: white, indenting: 0,
        width: 0, shortBitmap: NIL, tag: long, height: 0, longBitmap: NIL];
    csbPtr.wakeupMask ← Inline.BITOR[wakeVF, csbPtr.wakeupMask];
    pDCBBlank ← LOOPHOLE[@Environment.first64K[globalState]];
    pDCBReal ← LOOPHOLE[@Environment.first64K[globalState] + SIZE[DCB]];
    pDCBBlank↑ ← pDCBReal↑ ← dcb;
    pDCBReal.width ← width/Environment.bitsPerWord;
    pDCBReal.height ← height/2;
    END;
  -- Initialization

  Start: PUBLIC PROCEDURE = -- exported to (Head)StartChain
    BEGIN OPEN DOInputOutput;
    IF (controller ← GetNextController[utvfc, nullControllerNumber]) ~=
      nullControllerNumber THEN
      BEGIN
      inputReg: UTVFCInputRegister = Input[[controller: controller, register: 0]];
      controllerType ← utvfc;
      IF inputReg.bitClockRate = cslBitClockRate THEN
        BEGIN
        bitmapWidth ← width ← cslWordsPerLine*Environment.bitsPerWord;
        refreshRate ← cslRefreshRate;
        millisecondsPerTick ← cslMillisecondsPerTick;
        END;
      END
    ELSE
      IF (controller ← GetNextController[uibScb, nullControllerNumber]) ~=
        nullControllerNumber THEN
        BEGIN
        controllerType ← uibScb;
        width ← cslWordsPerLine*Environment.bitsPerWord;
```

```
        bitmapWidth ← torWordsPerLine*Environment.bitsPerWord;
        height ← torHeight;
        millisecondsPerTick ← torMillisecondsPerTick;
        END
      ELSE controllerType ← null;
    pagesForBitmap ←
      ((bitmapWidth/Environment.bitsPerWord)*height + Environment.wordsPerPage -
        1)/Environment.wordsPerPage;
    csbPtr.dcbChainHead ← pDCBNull; -- DCB's are in first64K
    RemainingHeads.Start[];
    END;

  END.
```

LOG

Time: February 6, 1980 3:55 PM. By: Gobbel Action: Create file from UserTerm
***inalImpl*
Time: February 8, 1980 12:29 PM By: McJones Action: Add start chaining
Time: February 25, 1980 1:46 PM By: McJones Action: Automatic determination
***of LF/CSL/Tor display*
Time: March 7, 1980 5:56 PM By: McJones Action: Wait for two frames in turnOff arm of
***cleanup procedure*
Time: June 26, 1980 11:04 AM By: McJones Action: OISProcessorFace = >ProcessorFac
***e; allow Disconnect in disconnected state; add cursorPosition, mousePosition*
Time: July 29, 1980 9:54 AM By: McJones Action: Add keyboard, hasBorder; split off M
***ouseFace*
Time: July 30, 1980 6:21 PM By: McJones Action: Add pagesForBitmap, GetBitBltTable
***s; buffered = >hasBuffer*

```
-- File: RS232CHeadFrontEndD0
-- LastEdited: August 5, 1980 5:18 PM  By: BRD

DIRECTORY
  Environment USING [Byte],
  MIOCInternalD0 USING [
    Cmds, DecrementOnCount, GetCount, Impl, IncrementOnCount, IOCB, IOCBPtr,
    maxLines, MakeMeANewInstance, LongIOCBPtr, QueueType],
  Process USING [InitializeCondition, MsecToTicks],
  ResidentHeap USING [first64K],
  RS232CCorrespondents USING [asciiByteSync, ebcdicByteSync, bitSync, failure],
  RS232CEnvironment USING [
    AutoRecognitionOutcome, CompletionHandle, PhysicalRecord,
    PhysicalRecordHandle],
  RS232CFace USING [DeviceStatus, ParamHandle, ParameterOutcome, TransferStatus],
  Zone USING [Base];

RS232CHeadFrontEndD0: MONITOR
  IMPORTS MIOCInternalD0, Process EXPORTS RS232CFace, RS232CEnvironment =
  BEGIN
  lineCount: CARDINAL;
  externalCall: ARRAY [0..MIOCInternalD0.maxLines) OF RECORD [
    impl: LONG POINTER TO MIOCInternalD0.Impl,
    cmds: LONG POINTER TO MIOCInternalD0.Cmds];
  CompletionHandle: PUBLIC TYPE = MIOCInternalD0.LongIOCBPtr;
  AbortInput: PUBLIC PROCEDURE [lineNumber: CARDINAL] =
    BEGIN externalCall[lineNumber].cmds.Abort[input]; END;

  AbortOutput: PUBLIC PROCEDURE [lineNumber: CARDINAL] =
    BEGIN externalCall[lineNumber].cmds.Abort[output]; END;

  AbortStatus: PUBLIC PROCEDURE [lineNumber: CARDINAL] =
    BEGIN externalCall[lineNumber].impl.AbortStatus[]; END;

  AutoRecognitionWait: PUBLIC PROCEDURE [lineNumber: CARDINAL]
    RETURNS [outcome: RS232CEnvironment.AutoRecognitionOutcome] =
    BEGIN
    timeout: CONDITION;
    newStat, stat: RS232CFace.DeviceStatus;
    completed: BOOLEAN ← FALSE;
    autoRecognitionProcess: PROCESS;
    AwaitAutoRecognitionCompletion: ENTRY PROCEDURE RETURNS [BOOLEAN] =
      BEGIN
      WAIT timeout;
      RETURN[~completed]; -- RETURNS TRUE is timeout occurred.

      END;
    DoAutoRecognition: PROCEDURE =
      BEGIN
      buffer: PACKED ARRAY [0..2] OF Environment.Byte;
      rec: RS232CEnvironment.PhysicalRecord ←
        [[NIL, 0, 0], [@buffer, 0, 2], [NIL, 0, 0]];
      byteCount: CARDINAL;
      status: RS232CFace.TransferStatus;
      outcome ← RS232CCorrespondents.failure;
      WHILE outcome = RS232CCorrespondents.failure DO
        [byteCount, status] ← TransferWait[lineNumber, Get[lineNumber, @rec]];
        IF status = aborted THEN EXIT;
        SELECT buffer[0] FROM
```



```
          125B =>
            IF byteCount = 2 THEN
              SELECT buffer[1] FROM
                106B => outcome ← RS232CCorrespondents.ebcdicByteSync;
                141B => outcome ← RS232CCorrespondents.asciiByteSync;
                ENDCASE;
          077B, 176B => outcome ← RS232CCorrespondents.bitSync;
          ENDCASE;
        ENDLOOP;
      SignalAutoRecognitionComplete[];
      END;
    SignalAutoRecognitionComplete: ENTRY PROCEDURE =
      BEGIN completed ← TRUE; NOTIFY timeout; END;
    stat ← GetStatus[lineNumber];
    UNTIL stat.dataSetReady DO
      newStat ← StatusWait[lineNumber, stat];
      IF newStat = stat THEN RETURN[RS232CCorrespondents.failure];
      stat ← newStat;
      ENDLOOP;
    Process.InitializeCondition[@timeout, Process.MsecToTicks[15000]];
    autoRecognitionProcess ← FORK DoAutoRecognition[];
    IF AwaitAutoRecognitionCompletion[] THEN AbortInput[lineNumber];
    JOIN autoRecognitionProcess;
    END;

  Get: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, rec: RS232CEnvironment.PhysicalRecordHandle]
    RETURNS [RS232CEnvironment.CompletionHandle] =
    BEGIN RETURN[externalCall[lineNumber].impl.Get[rec]]; END;

  GetLineCount: PUBLIC PROCEDURE RETURNS [numberOfLines: CARDINAL] =
    BEGIN RETURN[lineCount ← MIOCInternalD0.GetCount[]]; END;

  GetStatus: PUBLIC PROCEDURE [lineNumber: CARDINAL]
    RETURNS [stat: RS232CFace.DeviceStatus] =
    BEGIN
    RETURN[externalCall[lineNumber].impl.UpdateStatus[]];
    -- make certain that what we return is up to date

    END;

  Off: PUBLIC PROCEDURE [lineNumber: CARDINAL] =
    BEGIN
    MIOCInternalD0.DecrementOnCount[];
    externalCall[lineNumber].cmds.Cleanup[];
    externalCall[lineNumber].impl.Cleanup[];
    END;

  On: PUBLIC PROCEDURE [lineNumber: CARDINAL] =
    BEGIN
    [externalCall[lineNumber].cmds, externalCall[lineNumber].impl] ←
      MIOCInternalD0.MakeMeANewInstance[lineNumber];
    MIOCInternalD0.IncrementOnCount[];
    END;

  Put: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, rec: RS232CEnvironment.PhysicalRecordHandle]
    RETURNS [RS232CEnvironment.CompletionHandle] =
```

```
    BEGIN RETURN[externalCall[lineNumber].impl.Put[rec]]; END;

  ResetLine: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, paramHandle: RS232CFace.ParamHandle]
    RETURNS [outcome: RS232CFace.ParameterOutcome] =
    BEGIN RETURN[externalCall[lineNumber].cmds.ResetLine[paramHandle]]; END;

  SendBreak: PUBLIC PROCEDURE [lineNumber: CARDINAL] =
    BEGIN externalCall[lineNumber].cmds.SendBreak[]; END;

  SetParameters: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, paramHandle: RS232CFace.ParamHandle]
    RETURNS [outcome: RS232CFace.ParameterOutcome] =
    BEGIN RETURN[externalCall[lineNumber].cmds.SetParameters[paramHandle]]; END;

  StatusWait: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, stat: RS232CFace.DeviceStatus]
    RETURNS [newstat: RS232CFace.DeviceStatus] =
    BEGIN RETURN[externalCall[lineNumber].impl.StatusWait[stat]]; END;

  TransmitNow: PUBLIC PROCEDURE [lineNumber: CARDINAL, event: CompletionHandle]
    RETURNS [byteCount: CARDINAL, status: RS232CFace.TransferStatus] =
    BEGIN
    iocb: MIOCInternalD0.IOCBPtr ← Relative[ResidentHeap.first64K, event];
    IF event.type = input THEN NULL
    ELSE [byteCount, status] ← TransferWait[lineNumber, event];
    END;

  LongPointerDifference: TYPE = MACHINE DEPENDENT RECORD [
    lowHalf: Zone.Base RELATIVE POINTER, highHalf: CARDINAL];
  Relative: PROCEDURE [base, p: LONG POINTER]
    RETURNS [Zone.Base RELATIVE POINTER] =
    -- computes a RELATIVE POINTER with respect to the zoneBase of zone z.  No check is made t
**o see that p is within range of base
    INLINE BEGIN RETURN[LOOPHOLE[p - base, LongPointerDifference].lowHalf] END;

  TransferWait: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, event: RS232CEnvironment.CompletionHandle]
    RETURNS [byteCount: CARDINAL, status: RS232CFace.TransferStatus] =
    BEGIN
    [byteCount, status] ← externalCall[lineNumber].impl.TransferWait[event];
    END;

  END. -- RS232CHeadFrontEndD0
.LOG

Time: September 4, 1979 6:01 PM By: Bill Danielson Action: Created file
Time: September 12, 1979 3:17 PM By: Bill Danielson Action: Added AbortStatus
Time: October 2, 1979 2:04 PM By: Bill Danielson Action: Modified for MIOC implementation
Time: January 11, 1980 6:29 PM By: Bill Danielson Action: Moved cleanup routine to MIOCHardw
**are module
Time: January 17, 1980 12:57 PM By: Bill Danielson Action: Modified for new face definition.
Time: January 24, 1980 11:15 AM By: Bill Danielson Action: Fixed byte count on byte synchronou
**s lines.
Time: January 31, 1980 2:50 PM By: Bill Danielson Action: New buffer allocation scheme.
Time: March 3, 1980 12:13 PM By: Bill Danielson Action: Fix to prevent hang if no MIOC board.
Time: May 7, 1980 5:48 PM By: Bill Danielson Action: Changes for non-zero MDS and reduce MO
**NITOR lock time.
```



```
Time: May 21, 1980 11:15 AM By: Bill Danielson Action: Eliminate ERROR on zero length blocks.
Time: May 23, 1980 5:31 PM By: Bill Danielson Action: Muliple line support.
Time: June 25, 1980 1:43 PM By: Bill Danielson Action: Auto-recognition.
Time: July 21, 1980 4:00 PM By: Bill Danielson Action: Timeouts for autoRecognition.
Time: August 5, 1980 11:08 AM By: Bill Danielson Action: Change for EXPORTed type of RS232C
**Environment.CompletionHandle.
```

```
-- MIOCHardwareD0.mesa
-- Last edited: August 6, 1980 2:31 PM By: Danielson

DIRECTORY
  DeviceCleanup: FROM "DeviceCleanup" USING [Await, Item, Reason],
  -- Pilot mode --
  D0InputOutput: FROM "D0InputOutput" USING [
    ControllerNumber, GetNextController, IOPage, mioc, nullControllerNumber],
  Inline: FROM "Inline" USING [BITAND, BITOR, BITSHIFT],
  MIOCInternalD0: FROM "MIOCInternalD0" USING [
    Channel, ClockingType, ControllerStatusBlock, DialerLoadWord,
    DialerStatusWord, IOCBPtr, LoopbackType, maxLines, nullTimer,
    PollerCommandData, Timer, TimerAddress, TimerLoadWord, TimerMode,
    TimerModeWord, TTYPortLoadWord, TTYPortStatusWord],
  Mopcodes: FROM "Mopcodes" USING [zMISC],
  RS366Face: FROM "RS366Face" USING [GetStatusBits, SetStatusBits],
  Runtime: FROM "Runtime" USING [CallDebugger, GlobalFrame],
  RuntimeInternal: FROM "RuntimeInternal" USING [Codebase],
  SpecialSpace: FROM "SpecialSpace" USING [MakeCodeResident, MakeCodeSwappable],
  Zone: FROM "Zone" USING [Base, nil];

MIOCHardwareD0: MONITOR
  IMPORTS
    DeviceCleanup, D0InputOutput, Inline, Runtime, RuntimeInternal, SpecialSpace,
    Zone
  EXPORTS MIOCInternalD0, RS366Face =
  BEGIN
  -- various definitions
  currentMicrocode: PROGRAM ← NIL;
  maxLoops: CARDINAL ← 0;
  microcodeChanceCount: CARDINAL ← 20000;
  activeCount: CARDINAL ← 0;
  lineCount: CARDINAL ← 0;
  -- MIOC registers
  enableRegister: CARDINAL = 2;
  loadTimerRegister: CARDINAL = 3;
  readTimerRegister: CARDINAL = 2;
  loadDialerRegister: CARDINAL = 5;
  readDialerRegister: CARDINAL = 4;
  readTTYPortRegister: CARDINAL = 6;
  loadTTYPortRegister: CARDINAL = 13;
  clearRingLatchRegister: CARDINAL = 15;
  dialerLoadWord: ARRAY [0..MIOCInternalD0.maxLines) OF
    MIOCInternalD0.DialerLoadWord ← ALL[[0, FALSE, FALSE, FALSE, FALSE, 0]];
  disableTimerGate: PACKED ARRAY [0..MIOCInternalD0.maxLines) OF BOOLEAN ← ALL[
    FALSE];
  stopped: PACKED ARRAY [0..MIOCInternalD0.maxLines) OF BOOLEAN ← ALL[FALSE];
  initialized: PACKED ARRAY [0..MIOCInternalD0.maxLines) OF BOOLEAN ← ALL[FALSE];
  task: ARRAY [0..MIOCInternalD0.maxLines) OF D0InputOutput.ControllerNumber ←
    ALL[D0InputOutput.nullControllerNumber];
  DecrementOnCount: PUBLIC ENTRY PROCEDURE =
    BEGIN activeCount ← activeCount - 1; END;

  GetCount: PUBLIC PROCEDURE RETURNS [numberOfLines: CARDINAL] =
    BEGIN RETURN[lineCount]; END;

  GetCSBPointer: PUBLIC PROCEDURE [line: CARDINAL]
    RETURNS [csb: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock] =
```



```
    BEGIN
    IF line >= MIOCInternalD0.maxLines THEN RETURN[NIL]
    ELSE
      RETURN[
        IF task[line] = D0InputOutput.nullControllerNumber THEN NIL
        ELSE LOOPHOLE[@D0InputOutput.IOPage[task[line]]]];
    END;

  GetDialerCount: PUBLIC PROCEDURE RETURNS [dialerCount: CARDINAL] =
    -- Should check dialer bits --BEGIN RETURN[lineCount]; END;

  GetStatus: PUBLIC PROCEDURE [dialerNumber: CARDINAL]
    RETURNS [getStatusBits: RS366Face.GetStatusBits] =
    BEGIN
    dialerStatusWord: MIOCInternalD0.DialerStatusWord ← MIOCInput[
      task[dialerNumber], readDialerRegister];
    getStatusBits ←
      [powerIndication: ~dialerStatusWord.notDCEPowerIndicator,
        dataLineOccupied: ~dialerStatusWord.notDataLineOccupied,
        callOriginationStatus: ~dialerStatusWord.notCallOriginationStatus,
        presentNextDigit: ~dialerStatusWord.notPresentNextDigit,
        abandonCallAndRetry: ~dialerStatusWord.notAbandonCallAndRetry];
    END;

  IncrementOnCount: PUBLIC ENTRY PROCEDURE =
    BEGIN activeCount ← activeCount + 1; END;

  IssueChipCommand: PUBLIC ENTRY PROCEDURE [line: CARDINAL, command: UNSPECIFIED]
    RETURNS [result: UNSPECIFIED] =
    BEGIN
    ENABLE UNWIND => NULL;
    CSB: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock ← GetCSBPointer[
      line];
    IF CSB.pollerCommand.commandPart.command # noCommand THEN
      RestartMicrocode[line];
    CSB.pollerCommand ← command;
    DoLoadTimer[line, rs232cCommand, interruptOnTerminalCount, 18432];
    DoLoadTimer[line, rs232cCommand, squareWaveGenerator, 18432];
    GiveMicrocodeAChance[CSB];
    IF CSB.pollerCommand.commandPart.command # noCommand THEN
      RestartMicrocode[line];
    result ← CSB.pollerResults;
    END;

  LoadTimer: PUBLIC ENTRY PROCEDURE [
    line: CARDINAL, timer: MIOCInternalD0.Timer,
    timerMode: MIOCInternalD0.TimerMode, count: CARDINAL] =
    BEGIN DoLoadTimer[line, timer, timerMode, count]; END;

  LoadTTYPortRegister: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, ttyPortLoadWord: MIOCInternalD0.TTYPortLoadWord]
    RETURNS [stat: MIOCInternalD0.TTYPortStatusWord] =
    BEGIN
    MIOCOutput[task[lineNumber], loadTTYPortRegister, ttyPortLoadWord];
    stat.busy ← TRUE;
    UNTIL ~stat.busy DO
      stat ← MIOCInput[task[lineNumber], readTTYPortRegister]; ENDLOOP;
    END;
```

```
OverlayMicrocode: PUBLIC PROCEDURE [microcode: PROGRAM] RETURNS [BOOLEAN] =
  BEGIN
  IF activeCount > 1 AND currentMicrocode # microcode THEN RETURN[FALSE];
  currentMicrocode ← microcode;
  SpecialSpace.MakeCodeResident[currentMicrocode];
  -- When Pilot starts swapping frames, make frame data resident for the following call.
  LoadRamAndJump[RuntimeInternal.Codebase[currentMicrocode], FALSE];
  SpecialSpace.MakeCodeSwappable[currentMicrocode];
  RETURN[TRUE];
  END;

ResetRingLatch: PUBLIC PROCEDURE [line: CARDINAL] =
  BEGIN MIOCOutput[task[line], clearRingLatchRegister, 1]; END;

SetClocking: PUBLIC PROCEDURE [
  line: CARDINAL, clocking: MIOCInternalD0.ClockingType] =
  BEGIN
  dialerLoadWord[line].selectLocalTiming ←
    SELECT clocking FROM local => TRUE, ENDCASE --external-- => FALSE;
  MIOCOutput[task[line], loadDialerRegister, dialerLoadWord[line]];
  END;

SetLoopback: PUBLIC PROCEDURE [
  line: CARDINAL, loopback: MIOCInternalD0.LoopbackType] =
  BEGIN
  dialerLoadWord[line].loopbackEnable ←
    SELECT loopback FROM internal => TRUE, ENDCASE --external-- => FALSE;
  MIOCOutput[task[line], loadDialerRegister, dialerLoadWord[line]];
  END;

SetStatus: PUBLIC ENTRY PROCEDURE [
  dialerNumber: CARDINAL, setStatusBits: RS366Face.SetStatusBits] =
  BEGIN
  dialerLoadWord[dialerNumber] ←
    [unused: 0, callRequest: setStatusBits.callRequest,
      digitPresent: setStatusBits.digitPresent,
      selectLocalTiming: dialerLoadWord[dialerNumber].selectLocalTiming,
      loopbackEnable: dialerLoadWord[dialerNumber].loopbackEnable,
      digit: setStatusBits.digit];
  MIOCOutput[
    task[dialerNumber], loadDialerRegister, dialerLoadWord[dialerNumber]];
  END;

StartTransmitter: PUBLIC PROCEDURE [line: CARDINAL] =
  BEGIN
  startXmtr: MIOCInternalD0.PollerCommandData =
    [[channelA, startTransmitter, 0], 0];
  [] ← IssueChipCommand[line, startXmtr];
  END;

DoLoadTimer: PRIVATE PROCEDURE [
  line: CARDINAL, timer: MIOCInternalD0.Timer,
  timerMode: MIOCInternalD0.TimerMode, count: CARDINAL] =
  BEGIN
  timerWord: MIOCInternalD0.TimerModeWord ←
    [timerSelect: timer, mode: timerMode];
  timerLoadWord: MIOCInternalD0.TimerLoadWord ←
    [disableTimerGate: disableTimerGate[line], address: writeModeWord,
      unused: 0, data: 0];
```



```
  timerLoadWord ← Inline.BITOR[timerLoadWord, timerWord];
  LoadTimerRegister[line, timerLoadWord];
  timerLoadWord.address ←
    SELECT timer FROM
      rs232cClock => loadCounter0,
      rs232cCommand => loadCounter1,
      ENDCASE -- printerClock -- => loadCounter2;
  timerLoadWord.data ← Inline.BITAND[count, 377B];
  LoadTimerRegister[line, timerLoadWord];
  timerLoadWord.data ← Inline.BITSHIFT[count, -8];
  LoadTimerRegister[line, timerLoadWord];
  END;

DisableMIOC: PRIVATE PROCEDURE [line: CARDINAL] =
  BEGIN MIOCOutput[task[line], enableRegister, 0]; END;

EnableMIOC: PRIVATE PROCEDURE [line: CARDINAL] =
  BEGIN MIOCOutput[task[line], enableRegister, 1]; END;

GiveMicrocodeAChance: PRIVATE PROCEDURE [
  CSB: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock] =
  BEGIN
  i: CARDINAL ← 0;
  THROUGH [0..microcodeChanceCount] DO
    IF CSB.pollerCommand.commandPart.command = noCommand THEN RETURN;
    i ← i + 1; -- TEMP --
    IF i > maxLoops THEN maxLoops ← i; -- TEMP --
    ENDLOOP;
  END;

LoadTimerRegister: PRIVATE PROCEDURE [
  line: CARDINAL, timerLoadWord: MIOCInternalD0.TimerLoadWord] =
  BEGIN
  timerStatus: MACHINE DEPENDENT RECORD [
    busy: BOOLEAN,
    notRead: BOOLEAN,
    notWrite: BOOLEAN,
    notStrobe: BOOLEAN,
    address: MIOCInternalD0.TimerAddress,
    data: [0..377B]];
  MIOCOutput[task[line], loadTimerRegister, timerLoadWord];
  timerStatus.busy ← TRUE;
  UNTIL ~timerStatus.busy DO
    timerStatus ← MIOCInput[task[line], readTimerRegister]; ENDLOOP;
  END;

RestartMicrocode: PRIVATE PROCEDURE [line: CARDINAL] =
  BEGIN
  CSB: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock ← GetCSBPointer[
    line];
  DisableMIOC[line];
  -- [] ← OverlayMicrocode[currentMicrocode];
  DoLoadTimer[line, rs232cCommand, interruptOnTerminalCount, 18432];
  DoLoadTimer[line, rs232cCommand, squareWaveGenerator, 18432];
  SetTimerGate[line, TRUE];
  EnableMIOC[line];
  GiveMicrocodeAChance[CSB];
  IF CSB.pollerCommand.commandPart.command # noCommand THEN
```

```
    Runtime.CallDebugger["RS232CMicrocode Is Very Dead"L];
  END;

SetTimerGate: PRIVATE PROCEDURE [line: CARDINAL, gate: BOOLEAN] =
  BEGIN
  timerLoadWord: MIOCInternalD0.TimerLoadWord ←
    [disableTimerGate: ~gate, address: nop, unused: 0, data: 0];
  disableTimerGate[line] ← ~gate;
  LoadTimerRegister[line, timerLoadWord];
  END;

MIOCInput: PRIVATE PROCEDURE [
  task: D0InputOutput.ControllerNumber, register: UNSPECIFIED]
  RETURNS [data: UNSPECIFIED] = INLINE
  BEGIN RETURN[Input[task*16 + register]]; END;

MIOCOutput: PRIVATE PROCEDURE [
  task: D0InputOutput.ControllerNumber, register: UNSPECIFIED,
  data: UNSPECIFIED] = INLINE BEGIN Output[data, task*16 + register]; END;

Input: PRIVATE PROCEDURE [reg: UNSPECIFIED] RETURNS [data: UNSPECIFIED] =
  MACHINE CODE BEGIN Mopcodes.zMISC, 5; END;

Output: PRIVATE PROCEDURE [data: UNSPECIFIED, reg: UNSPECIFIED] = MACHINE CODE
  BEGIN Mopcodes.zMISC, 6; END;

LoadRamAndJump: PRIVATE PROCEDURE [LONG POINTER, BOOLEAN] = MACHINE CODE
  BEGIN Mopcodes.zMISC, 3; END;
-- Cleanup and Initialization procedures

InitializeRS232: PUBLIC PROCEDURE [line: CARDINAL] =
  BEGIN
  SetupCleanup[line];
  IF ~initialized[line] THEN BEGIN InitializeCSB[line]; EnableMIOC[line]; END;
  initialized[line] ← TRUE;
  END;

InitializePrinter: PUBLIC ENTRY PROCEDURE [line: CARDINAL, mask: UNSPECIFIED] =
  BEGIN
  CSB: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock ← GetCSBPointer[
    line];
  IF ~initialized[line] THEN BEGIN SetupCleanup[line]; InitializeCSB[line]; END;
  CSB.printerWakeup ← mask;
  IF ~initialized[line] THEN EnableMIOC[line];
  initialized[line] ← TRUE;
  END;

SetupCleanup: PRIVATE PROCEDURE [line: CARDINAL] =
  BEGIN
  item: DeviceCleanup.Item;
  reason: DeviceCleanup.Reason;
  miocEnableRegister: CARDINAL ← (task[line]*16) + enableRegister;
  DO
    reason ← DeviceCleanup.Await[@item];
    SELECT reason FROM
      turnOn =>
        BEGIN
        IF stopped[line] THEN Output[1, miocEnableRegister];
        stopped[line] ← FALSE;
```



```
        END;
      turnOff, kill =>
        BEGIN Output[0, miocEnableRegister]; stopped[line] ← TRUE; END;
      ENDCASE;
    ENDLOOP;
  END;

InitializeCSB: PRIVATE PROCEDURE [line: CARDINAL] =
  BEGIN
  CSB: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock ← GetCSBPointer[
    line];
  CSB↑ ←
    [printerWakeup: 0, notUsedYet1: ALL[0],
      pollerCommand: [[channelA, noCommand, 0], 0],
      pollerResults: [[channelA, noCommand, 0], 0],
      initialTimeoutCount: MIOCInternalD0.nullTimer,
      currentTimeoutCount: MIOCInternalD0.nullTimer, outputChain: Zone.nil,
      inputChain: Zone.nil, wakeup: 0, notUsedYet2: 0, reg5Data:, reg3Data:,
      notUsedYet4: 0, status: [0, FALSE, FALSE, FALSE], tables: [NIL, 0, 0]];
  DoLoadTimer[line, rs232cCommand, squareWaveGenerator, 18432];
  -- Start poller --
  SetTimerGate[line, TRUE];
  END;
-- MAIN PROGRAM --

line: CARDINAL;
currentTask: D0InputOutput.ControllerNumber ←
  D0InputOutput.nullControllerNumber;
SpecialSpace.MakeCodeResident[Runtime.GlobalFrame[GetCSBPointer]];
FOR line IN [0..MIOCInternalD0.maxLines) DO
  IF
    (task[line] ← D0InputOutput.GetNextController[
      D0InputOutput.mioc, currentTask]) # D0InputOutput.nullControllerNumber
    THEN BEGIN lineCount ← lineCount + 1; currentTask ← task[line]; END;
  ENDLOOP;
FOR line IN [0..lineCount/2) DO
  -- Flip entry to make board order correct --
  currentTask ← task[line];
  task[line] ← task[lineCount - 1 - line];
  task[lineCount - 1 - line] ← currentTask;
  ENDLOOP;
FOR line IN [0..lineCount) DO InitializeRS232[line]; ENDLOOP;
END. --MIOCHardwareD0
LOG
Time: January 11, 1980 6:11 PM By: Bill Danielson Action: Modified to use OISFace and D0Input
**Output to local MIOC board.
Time: January 11, 1980 6:29 PM By: Bill Danielson Action: Added cleanup routines.
Time: January 17, 1980 5:25 PM By: Bill Danielson Action: Converted to Pilot from Alto Emulation
**.
Time: January 22, 1980 5:02 PM By: Bill Danielson Action: Modified for new status and dataLost
**code.
Time: January 24, 1980 2:50 PM By: Bill Danielson Action: Combined AltoEm and Pilot code into
**single module.
Time: January 31, 1980 11:34 AM By: Bill Danielson Action: Removed export of Zone.nil from Pilo
**t version.
Time: February 7, 1980 4:27 PM By: McJones Action: Removed unnecessary deviceHandle varia
**ble in initialization.
Time: March 3, 1980 12:08 PM By: Danielson Action: Fix to prevent hang if no MIOC board.
```

Time: March 4, 1980 5:18 PM By: Danielson Action: Get CSB address from D0InputOutput.
Time: March 21, 1980 3:41 PM By: Danielson Action: Remove tables from global frame.
Time: March 24, 1980 11:53 AM By: Danielson Action: Reload microcode on debugger world swa
***ps.*
Time: May 8, 1980 1:50 PM By: Danielson Action: Added cleanup code for printer and removed A
***ltoEmulation code.*
Time: May 12, 1980 6:09 PM By: Artibee Action: Fix to Printer cleanup proc to enable MIOC at tur
***nOn.*
Time: May 21, 1980 11:47 AM By: Danielson Action: Replace task = 0 with task = D0InputOutput
***.nullController.*
Time: May 22, 1980 3:21 PM By: Danielson Action: Multiple line support.
Time: June 25, 1980 2:51 PM By: Artibee Action: Replace RS232CMicrocodeIsVeryDead ERROR
***with CallDebugger.*
Time: July 1, 1980 10:53 AM By: Danielson Action: Pagefix module earlier in start trap and load T
***TY microcode to try and eliminate bug during startup.*
Time: July 10, 1980 10:19 AM By: Danielson Action: Modifed to implement RS366 head with sepa
***rate input and output bits.*
Time: July 22, 1980 3:43 PM By: Danielson Action: TTY port code.
Time: August 6, 1980 2:31 PM By: Danielson Action: Fixed TTY port code to work with multiple lin
***es.*

```
-- MIOCCommImplD0.mesa
-- Last edited: September 8, 1980 3:57 PM By: Danielson

DIRECTORY
  ByteBlt USING [ByteBlt],
  Environment USING [Block, Byte, bytesPerWord],
  Inline USING [LongCOPY],
  MIOCInternalD0 USING [
    ControllerStatusBlock, GetCSBPointer, Impl, IOCB, IOCBPtr, IssueChipCommand,
    LatchBitType, LongIOCBPtr, PollerCommandData, QueueType, ReadReg0, ReadReg1,
    ResetRingLatch, ResetStatusCommand, StartTransmitter, State, WriteReg0],
  Process USING [
    MsecToTicks, Priority, SetPriority, SetTimeout, InitializeCondition, Yield],
  ProcessInternal USING [AllocateNakedCondition, DeallocateNakedCondition],
  ResidentHeap USING [MakeNode, FreeNode, first64K],
  RS232CEnvironment USING [
    CompletionHandle, LineType, PhysicalRecord, PhysicalRecordHandle],
  RS232CFace USING [DeviceStatus, ParamHandle, TransferStatus],
  Heap USING [Create, Delete, FreeNode, MakeNode],
  SpecialHeap USING [MakeResident, MakeSwappable],
  Runtime USING [CallDebugger, SelfDestruct],
  Zone USING [Base, nil, Status];

MIOCCommImplD0: MONITOR
  IMPORTS
    ByteBlt, Heap, Inline, MIOCInternalD0, Process, ProcessInternal, ResidentHeap,
    Runtime, SpecialHeap, Zone
  EXPORTS MIOCInternalD0, RS232CEnvironment =
  BEGIN
  -- various definitions
  CompletionHandle: PUBLIC TYPE = MIOCInternalD0.LongIOCBPtr;
  wakeUp: LONG POINTER TO CONDITION;
  -- gets naked notify when microcode does something interesting --
  pollerTimeout: CONDITION; -- poller timeout --
  statusChange: CONDITION; -- wakes up on status change --
  aMoment: CONDITION; -- short timeout for cleanup yielding --
  interrupt: PROCESS;
  poll: PROCESS;
  csmPriority: Process.Priority ← 3;
  -- Someday, this will come from a central defs file.
  line: CARDINAL;
  ending: BOOLEAN ← FALSE;
  inputIOCBCount, outputIOCBCount, statusWaitCount: CARDINAL ← 0;
  inputStart, outputStart: MIOCInternalD0.State;
  syncBuffer: PACKED ARRAY [0..12B] OF Environment.Byte ← ALL[252B];
  syncs: Environment.Block ← [@syncBuffer, 0, 0];
  fillBuffer: PACKED ARRAY [0..2B) OF Environment.Byte ← ALL[377B];
  -- Must be 0..2 due to Mesa 6 packing --
  fills: Environment.Block ← [@fillBuffer, 0, 1];
  lineType: RS232CEnvironment.LineType ← asynchronous;
  CSB: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock;
  bufferZone: UNCOUNTED ZONE;
  -- logSize: CARDINAL = 5;
  -- nextLogEntry: CARDINAL ← 0;
  -- logBuffer: ARRAY [0..logSize) OF LogBufferType ← ALL [[LOOPHOLE[0], NIL]];
  -- LogBufferType: TYPE = RECORD [
  -- iocb: MIOCInternalD0.IOCBPtr,
  -- bufferPtr: LONG POINTER TO ARRAY OF Environment.Byte];
```



```
  statusLocked, awaitingStatusLock: BOOLEAN ← FALSE;
  statusUnlocked: CONDITION;
  abortingStatus: BOOLEAN ← FALSE;
  readBreak: BOOLEAN ← TRUE;
  currentStatus: RS232CFace.DeviceStatus ←
    [FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE];
  inQueueStart: MIOCInternalD0.IOCBPtr ← Zone.nil;
  inQueueEnd: MIOCInternalD0.IOCBPtr ← Zone.nil;
  outQueueStart: MIOCInternalD0.IOCBPtr ← Zone.nil;
  outQueueEnd: MIOCInternalD0.IOCBPtr ← Zone.nil;
  impl: MIOCInternalD0.Impl ←
    [AbortStatus: AbortStatus, AbortQueue: AbortQueue, AbortTopIOCB: AbortTopIOCB,
      ClearLatchBit: ClearLatchBit, Cleanup: Cleanup, Get: Get,
      InitializeSyncBlock: InitializeSyncBlock, Put: Put,
      SetStartStates: SetStartStates, StatusWait: StatusWait,
      TransferWait: TransferWait, UpdateStatus: UpdateStatus];
  -- procedures (listed alphabetically)
  AbortQueue: PUBLIC PROCEDURE [type: MIOCInternalD0.QueueType] =
    BEGIN -- Abort all outstanding IOCBs, microcode has been stopped --
    iocb: MIOCInternalD0.IOCBPtr;
    SELECT type FROM
      input =>
        UNTIL inputIOCBCount = 0 DO
          -- inputIOCBCount not monitored since only read --
          IF (iocb ← CSB.inputChain) # Zone.nil THEN AbortTopIOCB[iocb];
          WaitAMoment[];
          ENDLOOP;
      output =>
        UNTIL outputIOCBCount = 0 DO
          -- outputIOCBCount not monitored since only read --
          IF (iocb ← CSB.outputChain) # Zone.nil THEN AbortTopIOCB[iocb];
          WaitAMoment[];
          ENDLOOP;
    ENDCASE;
    END;

  AbortStatus: PUBLIC ENTRY PROCEDURE =
    BEGIN -- Abort all outstanding StatusWaits --
    resetCSB: MIOCInternalD0.ResetStatusCommand ←
      [commandPart: [channelA, resetStatus, 0],
        dataPart: [dataLost: TRUE, event: TRUE]];
    resetStatusInterrupts: MIOCInternalD0.WriteReg0 ←
      [command: [channelA, writeChip, 0], data: [null, resetExternalStatus, 0]];
    abortingStatus ← TRUE;
    WHILE statusWaitCount > 0 DO BROADCAST statusChange; WAIT aMoment; ENDLOOP;
    currentStatus ← [FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE];
    [] ← MIOCInternalD0.IssueChipCommand[line, resetCSB];
    [] ← MIOCInternalD0.IssueChipCommand[line, resetStatusInterrupts];
    MIOCInternalD0.ResetRingLatch[line];
    abortingStatus ← FALSE;
    END;

  AbortTopIOCB: PUBLIC PROCEDURE [iocb: MIOCInternalD0.IOCBPtr] =
    BEGIN IF RemoveTopIOCBFromCSBQueue[iocb] THEN [] ← CompleteTheIOCB[iocb]; END;

  Cleanup: PUBLIC PROCEDURE =
    BEGIN -- Free buffers and kill all processes --
    ending ← TRUE;
```

```
  -- THROUGH [0..logSize) DO
  -- IF logBuffer[nextLogEntry].iocb # Zone.nil THEN
  -- BEGIN
  -- [] ← ResidentHeap.FreeNode[logBuffer[nextLogEntry].iocb];
  -- Heap.FreeNode[bufferZone, logBuffer[nextLogEntry].bufferPtr];
  -- logBuffer[nextLogEntry] ← [Zone.nil, NIL];
  -- END;
  -- nextLogEntry ← (nextLogEntry + 1) MOD logSize;
  -- ENDLOOP;
  CSB.wakeup ← 0;
  SpecialHeap.MakeSwappable[bufferZone];
  Heap.Delete[bufferZone];
  ForceInterrupt[];
  JOIN interrupt;
  ForcePollerWakeup[];
  JOIN poll;
  ProcessInternal.DeallocateNakedCondition[wakeUp];
  ending ← FALSE;
  IF line # 0 THEN Runtime.SelfDestruct[];
  END;

ClearLatchBit: PUBLIC ENTRY PROCEDURE [type: MIOCInternalD0.LatchBitType] =
  BEGIN
  SELECT type FROM
    breakDetected =>
      BEGIN currentStatus.breakDetected ← FALSE; NOTIFY pollerTimeout; END;
    dataLost =>
      BEGIN
      resetStatusDataLost: MIOCInternalD0.ResetStatusCommand ←
        [commandPart: [channelA, resetStatus, 0], dataPart: [dataLost: TRUE]];
      [] ← MIOCInternalD0.IssueChipCommand[line, resetStatusDataLost];
      END;
    ringHeard => BEGIN currentStatus.ringHeard ← FALSE; END;
    ENDCASE;
  END;

Get: PUBLIC PROCEDURE [rec: RS232CEnvironment.PhysicalRecordHandle]
  RETURNS [CompletionHandle] =
  BEGIN
  GetBlock: TYPE = RECORD [
    record: RS232CEnvironment.PhysicalRecord,
    buffer: PACKED ARRAY OF Environment.Byte];
  localStatus: Zone.Status;
  blockPtr: LONG POINTER TO GetBlock;
  dataLength: CARDINAL;
  iocb: MIOCInternalD0.IOCBPtr ← Zone.nil;
  iocbLong: MIOCInternalD0.LongIOCBPtr;
  -- The following block is defined so that the UNWIND can access local variables
  BEGIN
  ENABLE
    UNWIND =>
      BEGIN
      -- Also return data buffer if it's allocated
      -- Return IOCB and release monitor lock
      IF iocb # Zone.nil THEN [] ← ResidentHeap.FreeNode[iocb];
      END;
  -- Obtain an IOCB
  DO
    -- until EXIT
```



```
    [iocb, localStatus] ← ResidentHeap.MakeNode[SIZE[MIOCInternalD0.IOCB], a4];
    IF localStatus = okay THEN EXIT;
    Process.Yield[];
    ENDLOOP;
  iocbLong ← @ResidentHeap.first64K[iocb];
  -- Determine the size of and obtain the resident buffer required
  dataLength ←
    rec.header.stopIndexPlusOne - rec.header.startIndex +
      rec.body.stopIndexPlusOne - rec.body.startIndex +
      rec.trailer.stopIndexPlusOne - rec.trailer.startIndex;
  blockPtr ← Heap.MakeNode[
    bufferZone,
    (dataLength + 1)/Environment.bytesPerWord + SIZE[
      RS232CEnvironment.PhysicalRecord]];
  Inline.LongCOPY[
    from: rec, to: @blockPtr.record,
    nwords: SIZE[RS232CEnvironment.PhysicalRecord]];
  -- initialize IOCB
  LOOPHOLE[iocbLong, LONG POINTER]↑ ← 0;
  Inline.LongCOPY[
    from: iocbLong, to: iocbLong + 1, nwords: SIZE[MIOCInternalD0.IOCB] - 1];
  iocbLong.currentState ← inputStart;
  iocbLong.buffer ← @blockPtr.buffer;
  iocbLong.maxCount ← dataLength;
  iocbLong.type ← input;
  Process.InitializeCondition[@ResidentHeap.first64K[iocb].synch, 500];
  IncrementInputIOCBCount[];
  IF dataLength = 0 THEN ForceIOCBComplete[iocb] ELSE EnqueueIn[iocb];
  RETURN[iocbLong];
  END;
  END;

Initialize: PUBLIC PROCEDURE [currentLine: CARDINAL]
  RETURNS [implHandle: LONG POINTER TO MIOCInternalD0.Impl] =
  BEGIN
  line ← currentLine;
  bufferZone ← Heap.Create[
    initial: 2, increment: 2, largeNodeThreshold: Space.wordsPerPage*2];
  SpecialHeap.MakeResident[bufferZone];
  -- SpecialSpace.MakeCodeResident[Runtime.GlobalFrame[InterruptProcess]];
  CSB ← MIOCInternalD0.GetCSBPointer[line];
  currentStatus ← [FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE];
  [cv: wakeUp, mask: CSB.wakeup] ← ProcessInternal.AllocateNakedCondition[];
  Process.SetTimeout[wakeUp, Process.MsecToTicks[10000]];
  inputIOCBCount ← outputIOCBCount ← statusWaitCount ← 0;
  AbortStatus[];
  interrupt ← FORK InterruptProcess;
  poll ← FORK Poller;
  implHandle ← @impl;
  END;

InitializeSyncBlock: PUBLIC PROCEDURE [paramHandle: RS232CFace.ParamHandle] =
  BEGIN -- Set up SYN block and save current line type --
  i: INTEGER;
  syncs.stopIndexPlusOne ← paramHandle.syncCount + 3;
  FOR i IN [3..12B] DO syncBuffer[i] ← paramHandle.syncChar; ENDLOOP;
  lineType ← paramHandle.lineType;
  END;
```

```
MakeMeANewImpl: PUBLIC PROCEDURE [currentLine: CARDINAL]
  RETURNS [implHandle: LONG POINTER TO MIOCInternalD0.Impl] =
  BEGIN
  IF currentLine = 0 THEN RETURN[Initialize[currentLine]]
  ELSE
    BEGIN
    frame: POINTER TO FRAME[MIOCCommImplD0] ← NEW MIOCCommImplD0;
    RETURN[frame.Initialize[currentLine]];
    END;
  END;

Put: PUBLIC PROCEDURE [rec: RS232CEnvironment.PhysicalRecordHandle]
  RETURNS [CompletionHandle] =
  BEGIN
  localStatus: Zone.Status;
  bufferPtr: LONG POINTER;
  dataLength: CARDINAL;
  block: Environment.Block;
  iocbLong: MIOCInternalD0.LongIOCBPtr;
  iocb: MIOCInternalD0.IOCBPtr ← Zone.nil;
  -- The following block is defined so that the UNWIND can access local variables
  BEGIN
  ENABLE
    UNWIND =>
      BEGIN
      -- Also return data buffer if it's allocated
      -- Return IOCB and release monitor lock
      IF iocb # Zone.nil THEN [] ← ResidentHeap.FreeNode[iocb];
      END;
  -- Obtain an IOCB
  DO
    -- until EXIT
    [iocb, localStatus] ← ResidentHeap.MakeNode[SIZE[MIOCInternalD0.IOCB], a4];
    IF localStatus = okay THEN EXIT;
    Process.Yield[];
    ENDLOOP;
  iocbLong ← @ResidentHeap.first64K[iocb];
  -- Determine the size of and obtain the resident buffer required
  dataLength ←
    rec.header.stopIndexPlusOne - rec.header.startIndex +
      rec.body.stopIndexPlusOne - rec.body.startIndex +
      rec.trailer.stopIndexPlusOne - rec.trailer.startIndex +
      (IF lineType = byteSynchronous THEN syncs.stopIndexPlusOne + 1 ELSE 0);
  -- Copy all three of the clients buffers into one resident buffer
  IF dataLength # 0 THEN
    BEGIN
    bufferPtr ← Heap.MakeNode[
      bufferZone, (dataLength + 1)/Environment.bytesPerWord];
    block ← [bufferPtr, 0, dataLength];
    IF lineType = byteSynchronous THEN
      block.startIndex ←
        block.startIndex + ByteBlt.ByteBlt[to: block, from: syncs];
    block.startIndex ←
      block.startIndex + ByteBlt.ByteBlt[to: block, from: rec.header];
    block.startIndex ←
      block.startIndex + ByteBlt.ByteBlt[to: block, from: rec.body];
    block.startIndex ←
      block.startIndex + ByteBlt.ByteBlt[to: block, from: rec.trailer];
    IF lineType = byteSynchronous THEN
      block.startIndex ←
```



```
        block.startIndex + ByteBlt.ByteBlt[to: block, from: fills];
    END;
  -- initialize IOCB
  LOOPHOLE[iocbLong, LONG POINTER]↑ ← 0;
  Inline.LongCOPY[
    from: iocbLong, to: iocbLong + 1, nwords: SIZE[MIOCInternalD0.IOCB] - 1];
  iocbLong.currentState ← outputStart;
  iocbLong.buffer ← bufferPtr;
  iocbLong.maxCount ← dataLength;
  iocbLong.type ← output;
  Process.InitializeCondition[@iocbLong.synch, 500];
  IncrementOutputIOCBCount[];
  IF dataLength = 0 THEN ForceIOCBComplete[iocb] ELSE EnqueueOut[iocb];
  RETURN[iocbLong];
  END;
  END;

SetStartStates: PUBLIC PROCEDURE [
  input: MIOCInternalD0.State, output: MIOCInternalD0.State] =
  BEGIN inputStart ← input; outputStart ← output; END;

StatusWait: PUBLIC PROCEDURE [stat: RS232CFace.DeviceStatus]
  RETURNS [newstat: RS232CFace.DeviceStatus] =
  BEGIN
  IncrementStatusWaitCount[];
  newstat ← UpdateStatus[];
  WHILE ~abortingStatus AND stat = newstat DO
    AwaitStatusChange[]; newstat ← UpdateStatus[]; ENDLOOP;
  DecrementStatusWaitCount[];
  END;

TransferWait: PUBLIC PROCEDURE [event: CompletionHandle]
  RETURNS [byteCount: CARDINAL, status: RS232CFace.TransferStatus] =
  BEGIN
  bufferPtr: LONG ORDERED POINTER;
  block: Environment.Block;
  deviceStatus: RS232CFace.DeviceStatus;
  record: LONG POINTER TO RS232CEnvironment.PhysicalRecord;
  iocb: MIOCInternalD0.IOCBPtr;
  localByteCount: CARDINAL;
  localStatus: RS232CFace.TransferStatus ← success;
  bitChecksum: MIOCInternalD0.ReadReg1;
  iocb ← Relative[ResidentHeap.first64K, event];
  IF iocb = Zone.nil THEN Runtime.CallDebugger["IOCB Should Not be NIL"L];
  WaitForIOCBComplete[iocb];
  localByteCount ←
    event.offset - event.start -
      (IF lineType = byteSynchronous AND event.type = output THEN
       syncs.stopIndexPlusOne + 1 ELSE 0);
  localStatus ← event.transferCompletion;
  IF event.type = input AND localStatus = success THEN
    BEGIN
    SELECT lineType FROM
      asynchronous =>
        BEGIN
        IF event.checksum # 0 THEN localStatus ← checksumError;
        IF event.status.crcFramingError THEN localStatus ← asynchFramingError;
        END;
```

```
      byteSynchronous =>
        BEGIN IF event.checksum # 0 THEN localStatus ← checksumError; END;
      bitSynchronous =>
        BEGIN
        bitChecksum ← LOOPHOLE[event.checksum, MIOCInternalD0.ReadReg1];
        IF bitChecksum.crcFramingError THEN localStatus ← checksumError;
        END;
      ENDCASE;
    IF event.status.receiverOverrun THEN localStatus ← dataLost;
    IF event.status.parityError THEN localStatus ← parityError;
    END;
  bufferPtr ← LOOPHOLE[event.buffer];
  -- for input operations, this value is overwritten below
  -- For inputs, copy from resident buffer to client's buffers and check CRC
  IF event.type = input THEN
    BEGIN
    deviceStatus ← currentStatus;
    IF
      (localStatus = success AND
        (deviceStatus.dataLost OR deviceStatus.breakDetected)) THEN
      localStatus ← deviceError;
    block ← [bufferPtr, 0, localByteCount];
    record ← bufferPtr ← bufferPtr - SIZE[RS232CEnvironment.PhysicalRecord];
    block.startIndex ←
      block.startIndex + ByteBlt.ByteBlt[to: record.header, from: block];
    block.startIndex ←
      block.startIndex + ByteBlt.ByteBlt[to: record.body, from: block];
    block.startIndex ←
      block.startIndex + ByteBlt.ByteBlt[to: record.trailer, from: block];
    IF block.startIndex # block.stopIndexPlusOne THEN
      Runtime.CallDebugger["We Should Have Moved Exactly Everything"L];
    END;
  FreeBuffers[iocb, bufferPtr];
  RETURN[localByteCount, localStatus];
  END;

UpdateStatus: PUBLIC PROCEDURE RETURNS [stat: RS232CFace.DeviceStatus] =
  BEGIN
  gotBreak: BOOLEAN ← FALSE;
  readReg0: MIOCInternalD0.PollerCommandData ← [[channelA, readChip, 0], 0];
  reg0: MIOCInternalD0.ReadReg0 ← MIOCInternalD0.IssueChipCommand[
    line, readReg0];
  resetStatusEvent: MIOCInternalD0.ResetStatusCommand ←
    [commandPart: [channelA, resetStatus, 0], dataPart: [event: TRUE]];
  GetStatusLock[];
  [] ← MIOCInternalD0.IssueChipCommand[line, resetStatusEvent];
  IF readBreak AND reg0.break AND lineType = asynchronous THEN
    BEGIN gotBreak ← TRUE; readBreak ← FALSE; END;
  IF ~readBreak AND ~reg0.break AND ~currentStatus.breakDetected THEN
    readBreak ← TRUE;
  stat ←
    [dataLost: CSB.status.dataLost,
      breakDetected: currentStatus.breakDetected OR gotBreak,
      clearToSend: reg0.clearToSend, dataSetReady: ~reg0.notDataSetReady,
      carrierDetect: ~reg0.carrierDetect,
      ringHeard: currentStatus.ringHeard OR reg0.ringIndicatorLatch,
      ringIndicator: reg0.ringIndicatorLatch];
  IF reg0.ringIndicatorLatch THEN MIOCInternalD0.ResetRingLatch[line];
```



```
  BroadcastStatusChange[stat];
  ReleaseStatusLock[];
  END;

AwaitPollerTimeout: ENTRY PROCEDURE = BEGIN WAIT pollerTimeout; END;

AwaitStatusChange: ENTRY PROCEDURE = BEGIN WAIT statusChange; END;

BroadcastStatusChange: ENTRY PROCEDURE [stat: RS232CFace.DeviceStatus] =
  BEGIN
  IF stat # currentStatus THEN
    BEGIN currentStatus ← stat; BROADCAST statusChange; END;
  END;

CompleteTheIOCB: ENTRY PROCEDURE [iocb: MIOCInternalD0.IOCBPtr]
  RETURNS [nextIOCB: MIOCInternalD0.IOCBPtr] =
  BEGIN
  -- The IOCB has already been removed from the CSB chain by the microcode.
  -- For now, all IOCB completions cause a NOTIFY
  ResidentHeap.first64K[iocb].completed ← TRUE;
  nextIOCB ← ResidentHeap.first64K[iocb].next;
  NOTIFY ResidentHeap.first64K[iocb].synch;
  SELECT ResidentHeap.first64K[iocb].type FROM
    input =>
      IF inQueueStart = iocb THEN
        BEGIN
        inQueueStart ← nextIOCB;
        IF nextIOCB = Zone.nil THEN inQueueEnd ← Zone.nil;
        END;
    output =>
      IF outQueueStart = iocb THEN
        BEGIN
        outQueueStart ← nextIOCB;
        IF nextIOCB = Zone.nil THEN outQueueEnd ← Zone.nil;
        END;
    ENDCASE;
  RETURN;
  END;

DecrementStatusWaitCount: ENTRY PROCEDURE =
  BEGIN statusWaitCount ← statusWaitCount - 1; END;

EnqueueIn: ENTRY PROCEDURE [iocb: MIOCInternalD0.IOCBPtr] =
  BEGIN
  IF inQueueEnd # Zone.nil THEN ResidentHeap.first64K[inQueueEnd].next ← iocb;
  inQueueEnd ← iocb;
  IF inQueueStart = Zone.nil THEN inQueueStart ← iocb;
  IF CSB.inputChain = Zone.nil AND ResidentHeap.first64K[iocb].currentState #
    377B THEN CSB.inputChain ← iocb;
  END;

EnqueueOut: ENTRY PROCEDURE [iocb: MIOCInternalD0.IOCBPtr] =
  BEGIN
  IF outQueueEnd # Zone.nil THEN ResidentHeap.first64K[outQueueEnd].next ← iocb;
  outQueueEnd ← iocb;
  IF outQueueStart = Zone.nil THEN outQueueStart ← iocb;
  IF CSB.outputChain = Zone.nil AND ResidentHeap.first64K[iocb].currentState #
    377B THEN
    BEGIN CSB.outputChain ← iocb; MIOCInternalD0.StartTransmitter[line]; END;
```

```
  END;

ForceIOCBComplete: ENTRY PROCEDURE [iocb: MIOCInternalD0.IOCBPtr] =
  BEGIN
  ResidentHeap.first64K[iocb].completed ← TRUE;
  NOTIFY ResidentHeap.first64K[iocb].synch;
  END;

ForcePollerWakeup: ENTRY PROCEDURE = BEGIN NOTIFY pollerTimeout; END;

ForceInterrupt: ENTRY PROCEDURE = BEGIN NOTIFY wakeUp↑; END;

FreeBuffers: ENTRY PROCEDURE [
  iocb: MIOCInternalD0.IOCBPtr, bufferPtr: LONG POINTER] =
  BEGIN
  SELECT ResidentHeap.first64K[iocb].type FROM
    input => inputIOCBCount ← inputIOCBCount - 1;
    output => outputIOCBCount ← outputIOCBCount - 1;
    ENDCASE;
  -- IF logBuffer[nextLogEntry].iocb # Zone.nil THEN
  -- BEGIN
  -- [] ← ResidentHeap.FreeNode[logBuffer[nextLogEntry].iocb];
  -- Heap.FreeNode[bufferZone, logBuffer[nextLogEntry].bufferPtr];
  -- END;
  -- logBuffer[nextLogEntry] ← [iocb, bufferPtr];
  -- nextLogEntry ← (nextLogEntry + 1) MOD logSize;
  [] ← ResidentHeap.FreeNode[iocb];
  Heap.FreeNode[bufferZone, bufferPtr];
  END;

GetStatusLock: ENTRY PROCEDURE =
  BEGIN
  WHILE statusLocked DO awaitingStatusLock ← TRUE; WAIT statusUnlocked; ENDLOOP;
  statusLocked ← TRUE;
  END;

IncrementInputIOCBCount: ENTRY PROCEDURE =
  BEGIN inputIOCBCount ← inputIOCBCount + 1; END;

IncrementOutputIOCBCount: ENTRY PROCEDURE =
  BEGIN outputIOCBCount ← outputIOCBCount + 1; END;

IncrementStatusWaitCount: ENTRY PROCEDURE =
  BEGIN statusWaitCount ← statusWaitCount + 1; END;

InterruptProcess: PROCEDURE =
  BEGIN
  iocb: MIOCInternalD0.IOCBPtr;
  Process.SetPriority[csmPriority];
  DO
    -- Later, consider if channel code should ever go away, once started.
    WaitForWakeUp[];
    IF ending THEN RETURN;
    IF CSB.status.event OR CSB.status.dataLost THEN ForcePollerWakeup[];
    -- Issue NOTIFY's on completed output IOCBs
    iocb ← outQueueStart;
    WHILE iocb # Zone.nil DO
      -- As long as there are completed output IOCBs
      BEGIN
```



```
      IF ResidentHeap.first64K[iocb].currentState # 377B THEN GO TO outputDone;
      [iocb] ← CompleteTheIOCB[iocb];
      END;
    REPEAT outputDone => NULL;
    ENDLOOP;
  -- Issue NOTIFY's on completed input IOCBs
  iocb ← inQueueStart;
  WHILE iocb # Zone.nil DO
    -- As long as there are completed input IOCBs
    BEGIN
    IF ResidentHeap.first64K[iocb].currentState # 377B THEN GO TO inputDone;
    [iocb] ← CompleteTheIOCB[iocb];
    END;
  REPEAT inputDone => NULL;
  ENDLOOP;
ENDLOOP;
END;

LongPointerDifference: TYPE = MACHINE DEPENDENT RECORD [
  lowHalf: Zone.Base RELATIVE POINTER, highHalf: CARDINAL];
Poller: PROCEDURE =
  BEGIN
  DO AwaitPollerTimeout[]; IF ending THEN RETURN; [] ← UpdateStatus[]; ENDLOOP;
  END;

Relative: PROCEDURE [base, p: LONG POINTER]
  RETURNS [Zone.Base RELATIVE POINTER] =
  -- computes a RELATIVE POINTER with respect to the zoneBase of zone z.  No check is made t
**o see that p is within range of base
  INLINE BEGIN RETURN[LOOPHOLE[p - base, LongPointerDifference].lowHalf] END;

ReleaseStatusLock: ENTRY PROCEDURE =
  BEGIN
  statusLocked ← FALSE;
  IF awaitingStatusLock THEN
    BEGIN NOTIFY statusUnlocked; awaitingStatusLock ← FALSE; END;
  END;

RemoveTopIOCBFromCSBQueue: PRIVATE ENTRY PROCEDURE [
  iocb: MIOCInternalD0.IOCBPtr] RETURNS [removedFromQueue: BOOLEAN] =
  BEGIN
  IF iocb = CSB.inputChain THEN
    BEGIN
    ResidentHeap.first64K[iocb].transferCompletion ← aborted;
    CSB.inputChain ← ResidentHeap.first64K[CSB.inputChain].next;
    RETURN[TRUE];
    END;
  IF iocb = CSB.outputChain THEN
    BEGIN
    ResidentHeap.first64K[iocb].transferCompletion ← aborted;
    CSB.outputChain ← ResidentHeap.first64K[CSB.outputChain].next;
    RETURN[TRUE];
    END;
  RETURN[FALSE];
  END;

WaitAMoment: ENTRY PROCEDURE = BEGIN WAIT aMoment; END;

WaitForIOCBComplete: ENTRY PROCEDURE [iocb: MIOCInternalD0.IOCBPtr] =
```

```
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    UNTIL ResidentHeap.first64K[iocb].completed DO
      WAIT ResidentHeap.first64K[iocb].synch; ENDLOOP;
    END;

  WaitForWakeUp: ENTRY PROCEDURE = BEGIN WAIT wakeUp; END;
  --Main program
  -- SpecialSpace.MakeCodeResident[Runtime.GlobalFrame[InterruptProcess]];

  Process.SetTimeout[@aMoment, 1]; -- 50 millseconds
  Process.SetTimeout[@pollerTimeout, 100]; -- 5 seconds
  Process.SetTimeout[@statusChange, 200]; -- 10 seconds
  Process.SetTimeout[@statusUnlocked, 100]; -- 5 seconds

  END. -- MIOCCommImplD0
LOG
Time: September 26, 1979 3:13 PM By: Bill Danielson Action: Created file
Time: December 17, 1979 6:20 PM By: Bill Danielson Action: Moved DeviceStatus code
**here.
Time: January 17, 1980 3:13 PM By: Bill Danielson Action: Changes for new face.
Time: January 21, 1980 6:25 PM By: Bill Danielson Action: New status code.
Time: January 22, 1980 2:44 PM By: Bill Danielson Action: Made changes for RS232
**CEnvironment.
Time: January 23, 1980 7:12 PM By: Bill Danielson Action: Rewrote status code to no
**t hold monitor lock for long periods of time.
Time: March 3, 1980 12:00 PM By: Bill Danielson Action: Modified to remove references to IO
**CS.
Time: March 5, 1980 3:45 PM By: Bill Danielson Action: Added support for TTY mode.
Time: March 19, 1980 1:52 PM By: Bill Danielson Action: Added code for TTY timeout.
Time: May 21, 1980 11:20 AM By: Bill Danielson Action: Remove ERROR on zero length fram
**e.
Time: May 23, 1980 10:38 AM By: Bill Danielson Action: Implement multiple lines.
Time: June 25, 1980 3:02 PM By: Mary Artibee Action: Replace ERRORs in TransferWait wit
**h CallDebugger.
Time: July 8, 1980 1:49 PM By: Bill Danielson Action: Turn off logging and fix PACKED AR
**RAY bug.
Time: July 23, 1980 4:55 PM By: Bill Danielson Action: Add code for new enable/disable xm
**tr.
Time: July 28, 1980 5:34 PM By: Bill Danielson Action: Remove RS232CHeap.
Time: August 4, 1980 12:57 PM By: Bill Danielson Action: New status code.
Time: August 7, 1980 5:13 PM By: Bill Danielson Action: Free condition variable after done us
**ing itl.
Time: August 29, 1980 6:05 PM By: McJones Action: Convert to {Allocate,Deallocate}Nak
**edCondition
Time: September 8, 1980 3:58 PM By: Danielson Action: Free condition variable co
**rrectly.
```

```
-- MIOCCommCmdsD0.mesa
-- Last edited: August 12, 1980 2:28 PM By: BRD

DIRECTORY
  Environment USING [Byte, wordsPerPage],
  Heap USING [Create, Delete, FreeNode, MakeNode],
  Inline USING [BITAND, BITOR, BITSHIFT, BITXOR, LongCOPY, LowHalf, HighHalf],
  MIOCInternalD0 USING [
    bps50, bps75, bps110, bps134p5, bps150, bps300, bps600, bps1200, bps2400,
    bps3600, bps4800, bps7200, bps9600, ControllerStatusBlock, Cmds,
    GetCSBPointer, Impl, InputEvent, IOCBPtr, IssueChipCommand, LoadTimer,
    MakeMeANewImpl, nullTimer, OutputEvent, OverlayMicrocode, PollerCommandData,
    QueueType, ResetStatusCommand, RS232CAsyncMicrocodeD0, RS232CByteMicrocodeD0,
    RS232CBitMicrocodeD0, RS232CTtyMicrocodeD0, SetClocking, State, System6Init,
    Tables, TimerSpeedConstant, WriteReg0, WriteReg1, WriteReg2, WriteReg3,
    WriteReg4, WriteReg5, WriteReg6, WriteReg7, X800Init, X850Init],
  Process USING [DisableTimeout, MsecToTicks, SetTimeout],
  ResidentHeap USING [first64K],
  RS232CCorrespondents USING [
    cmcII, oisSystemElement, system6, ttyHost, xerox800, xerox850],
  RS232CFace USING [ParamHandle, ParameterOutcome, ParameterRecord],
  Runtime USING [SelfDestruct],
  SpecialHeap USING [MakeResident, MakeSwappable],
  Zone USING [nil];

MIOCCommCmdsD0: MONITOR
  IMPORTS Heap, Inline, MIOCInternalD0, Process, Runtime, SpecialHeap, Zone
  EXPORTS MIOCInternalD0 =
  BEGIN
  -- various definitions
  line: CARDINAL;
  impl: LONG POINTER TO MIOCInternalD0.Impl;
  CSB: LONG POINTER TO MIOCInternalD0.ControllerStatusBlock;
  cmds: MIOCInternalD0.Cmds ←
    [Abort: Abort, Cleanup: Cleanup, ResetLine: ResetLine,
      SetParameters: SetParameters, SendBreak: SendBreak];
  autoRecognitionParameters: RS232CFace.ParameterRecord;
  currentParameters: RS232CFace.ParameterRecord;
  tablesZone: UNCOUNTED ZONE;
  tables: LONG POINTER ← NIL;
  globalOutcome: RS232CFace.ParameterOutcome;
  breakTimeout: CONDITION;
  power, val, crc, shift1, shift2, magic, bit: UNSPECIFIED;
  rev: BOOLEAN;
  globalSet, majorChange, minorChange: BOOLEAN ← FALSE;
  writeReg1: MIOCInternalD0.WriteReg1;
  writeReg3: MIOCInternalD0.WriteReg3;
  writeReg4: MIOCInternalD0.WriteReg4;
  writeReg5: MIOCInternalD0.WriteReg5;
  writeReg6: MIOCInternalD0.WriteReg6;
  writeReg7: MIOCInternalD0.WriteReg7;
  frameTimer: CONDITION; -- times out long input frames --
  frame: PROCESS;
  ending: BOOLEAN;
  Abort: PUBLIC PROCEDURE [queue: MIOCInternalD0.QueueType] =
    BEGIN Suspend[queue]; impl.AbortQueue[queue]; Restart[queue]; END;

  Cleanup: PUBLIC PROCEDURE =
```



```
    BEGIN
    ending ← TRUE;
    FreeTables[];
    ForceFrameTimerWakeup[];
    JOIN frame;
    IF line # 0 THEN Runtime.SelfDestruct[];
    END;

  Initialize: PUBLIC PROCEDURE [currentLine: CARDINAL]
    RETURNS [
      cmdHandle: LONG POINTER TO MIOCInternalD0.Cmds,
      implHandle: LONG POINTER TO MIOCInternalD0.Impl] =
    BEGIN
    statusAffectsVector: MIOCInternalD0.WriteReg1 ←
      [command: [channelB, writeChip, 1],
        data: [FALSE, FALSE, FALSE, disabled, TRUE, FALSE, FALSE]];
    interruptVector: MIOCInternalD0.WriteReg2 ←
      [command: [channelB, writeChip, 2], interruptVector: 0];
    reset: MIOCInternalD0.WriteReg0 ←
      [command: [channelA, writeChip, 0], data: [null, channelReset, 0]];
    writeReg1 ←
      [command: [channelA, writeChip, 1],
        data: [FALSE, FALSE, FALSE, allWithParity, FALSE, TRUE, TRUE]];
    writeReg3 ←
      [command: [channelA, writeChip, 3],
        data: [lengthIs8Bits, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE]];
    writeReg4 ←
      [command: [channelA, writeChip, 4],
        data: [times16, externalSync, two, none]];
    writeReg5 ←
      [command: [channelA, writeChip, 5],
        data: [FALSE, lengthIs8Bits, FALSE, FALSE, crcSDLC, FALSE, TRUE]];
    writeReg6 ← [command: [channelA, writeChip, 6], syncChar: 62B];
    writeReg7 ← [command: [channelA, writeChip, 7], syncChar: 62B];
    line ← currentLine;
    CSB ← MIOCInternalD0.GetCSBPointer[line];
    ending ← FALSE;
    [] ← MIOCInternalD0.IssueChipCommand[line, reset];
    [] ← MIOCInternalD0.IssueChipCommand[line, interruptVector];
    [] ← MIOCInternalD0.IssueChipCommand[line, statusAffectsVector];
    implHandle ← impl ← MIOCInternalD0.MakeMeANewImpl[line];
    Process.DisableTimeout[@frameTimer];
    frame ← FORK FrameTimerProcess;
    cmdHandle ← @cmds;
    END;

  MakeMeANewInstance: PUBLIC PROCEDURE [currentLine: CARDINAL]
    RETURNS [
      cmdHandle: LONG POINTER TO MIOCInternalD0.Cmds,
      implHandle: LONG POINTER TO MIOCInternalD0.Impl] =
    BEGIN
    IF currentLine = 0 THEN [cmdHandle, implHandle] ← Initialize[currentLine]
    ELSE
      BEGIN
      frame: POINTER TO FRAME[MIOCCommCmdsD0] ← NEW MIOCCommCmdsD0;
      [cmdHandle, implHandle] ← frame.Initialize[currentLine];
      END;
    END;
```

```
ResetLine: PUBLIC PROCEDURE [paramHandle: RS232CFace.ParamHandle]
  RETURNS [outcome: RS232CFace.ParameterOutcome] =
  BEGIN
  Suspend[input];
  Suspend[output];
  impl.AbortQueue[input];
  impl.AbortQueue[output];
  globalSet ← TRUE;
  outcome ← SetParameters[paramHandle];
  Restart[input];
  Restart[output];
  END;

SetParameters: PUBLIC ENTRY PROCEDURE [paramHandle: RS232CFace.ParamHandle]
  RETURNS [outcome: RS232CFace.ParameterOutcome] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  disableInterrupts: MIOCInternalD0.WriteReg1 ←
    [command: [channelA, writeChip, 1],
      data: [FALSE, FALSE, FALSE, disabled, FALSE, FALSE, FALSE]];
  IF paramHandle.lineType = autoRecognition THEN
    paramHandle ← BuildAutoRecognitionParameterRecord[paramHandle];
  IF
    ((currentParameters.lineType # paramHandle.lineType) OR
      (currentParameters.correspondent # paramHandle.correspondent)) THEN
    BEGIN
    globalSet ← TRUE;
    [] ← MIOCInternalD0.IssueChipCommand[line, disableInterrupts];
    END;
  globalOutcome ← success;
  IF globalSet OR currentParameters.lineType # paramHandle.lineType THEN
    BEGIN IF ~SetLineType[paramHandle] THEN globalOutcome ← unimplemented; END;
  IF globalSet OR currentParameters.lineSpeed # paramHandle.lineSpeed THEN
    SetLineSpeed[paramHandle];
  IF globalSet OR currentParameters.stopBits # paramHandle.stopBits AND
    paramHandle.lineType = asynchronous THEN SetStopBits[paramHandle];
  IF globalSet OR currentParameters.parity # paramHandle.parity THEN
    SetParity[paramHandle];
  IF globalSet OR currentParameters.correspondent # paramHandle.correspondent
    THEN SetCorrespondent[paramHandle];
  IF globalSet OR currentParameters.charLength # paramHandle.charLength THEN
    SetCharLength[paramHandle];
  IF globalSet OR currentParameters.syncCount # paramHandle.syncCount AND
    paramHandle.lineType = byteSynchronous THEN SetSyncCount[paramHandle];
  IF globalSet OR currentParameters.syncChar # paramHandle.syncChar AND
    paramHandle.lineType = byteSynchronous THEN SetSyncChar[paramHandle];
  impl.InitializeSyncBlock[paramHandle];
  IF globalSet OR currentParameters.frameTimeout # paramHandle.frameTimeout THEN
    SetFrameTimeout[paramHandle];
  IF globalSet OR currentParameters.dataTerminalReady #
    paramHandle.dataTerminalReady THEN SetDataTerminalReady[paramHandle];
  IF globalSet OR currentParameters.requestToSend # paramHandle.requestToSend
    THEN SetRequestToSend[paramHandle];
  IF paramHandle.resetBreakDetected THEN impl.ClearLatchBit[breakDetected];
  IF paramHandle.resetRingHeard THEN impl.ClearLatchBit[ringHeard];
  IF paramHandle.resetDataLost THEN impl.ClearLatchBit[dataLost];
  globalSet ← FALSE;
  SetRegistersNow[paramHandle];
```



```
  Inline.LongCOPY[
    from: paramHandle, to: @currentParameters,
    nwords: SIZE[RS232CFace.ParameterRecord]];
  RETURN[globalOutcome];
  END;

SendBreak: PUBLIC ENTRY PROCEDURE =
  BEGIN
  setRegister5: MIOCInternalD0.PollerCommandData ←
    [commandPart: [channelA, loadRegister5, 0], dataPart: 0];
  BEGIN
  ENABLE
    UNWIND =>
      BEGIN
      writeReg5.data.sendBreak ← FALSE;
      [] ← MIOCInternalD0.IssueChipCommand[line, setRegister5];
      END;
  IF currentParameters.lineType # asynchronous THEN RETURN;
  writeReg5.data.sendBreak ← TRUE;
  [] ← MIOCInternalD0.IssueChipCommand[line, setRegister5];
  WAIT breakTimeout;
  writeReg5.data.sendBreak ← FALSE;
  [] ← MIOCInternalD0.IssueChipCommand[line, setRegister5];
  END;
  END;

AllocateTables: PRIVATE PROCEDURE
  RETURNS [LONG POINTER TO MIOCInternalD0.Tables] =
  BEGIN
  tableSizeInPages: CARDINAL =
    (SIZE[MIOCInternalD0.Tables] +
      Environment.wordsPerPage)/Environment.wordsPerPage;
  alignedAddress: LONG POINTER;
  IF tables = NIL THEN
    BEGIN
    tablesZone ← Heap.Create[
      initial: tableSizeInPages, largeNodeThreshold: tableSizeInPages];
    SpecialHeap.MakeResident[tablesZone];
    tables ← Heap.MakeNode[tablesZone, SIZE[MIOCInternalD0.Tables] + 1];
    END;
  alignedAddress ←
    IF Inline.BITAND[Inline.LowHalf[tables], 1] = 0 THEN tables ELSE tables + 1;
  CSB.tables ←
    [lowHalf: Inline.LowHalf[alignedAddress],
      highHalf: Inline.HighHalf[alignedAddress],
      highHalfPlusOne: Inline.HighHalf[alignedAddress] + 1];
  RETURN[LOOPHOLE[alignedAddress]];
  END;

BuildAutoRecognitionParameterRecord: PRIVATE INTERNAL PROCEDURE [
  paramHandle: RS232CFace.ParamHandle]
  RETURNS [newParamHandle: RS232CFace.ParamHandle] =
  BEGIN
  autoRecognitionParameters ←
    [lineType: autoRecognition, correspondent: RS232CCorrespondents.ttyHost,
      lineSpeed: bps1200, parity: none, stopBits: 1, charLength: 8,
      syncCount: 0, syncChar: 0B, frameTimeout: 0,
      requestToSend: paramHandle.requestToSend,
      dataTerminalReady: paramHandle.dataTerminalReady, resetRingHeard: FALSE,
```

```
    resetBreakDetected: FALSE, resetDataLost: FALSE];
  globalSet ← TRUE;
  RETURN[@autoRecognitionParameters];
  END;

BuildChecksumTables: PRIVATE PROCEDURE [
  pointer: LONG POINTER TO MIOCInternalD0.Tables] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..255] DO
    crc ← 0;
    val ← Inline.BITSHIFT[i, shift1];
    FOR power IN [0..7] DO
      IF Inline.BITAND[val, bit] = 0 THEN val ← Inline.BITSHIFT[val, shift2]
      ELSE
        BEGIN
        crc ← Inline.BITXOR[crc, Inline.BITSHIFT[magic, shift2*(7 - power)]];
        val ← Inline.BITXOR[magic, Inline.BITSHIFT[val, shift2]];
        END;
      ENDLOOP;
    IF rev THEN
      crc ← Inline.BITOR[Inline.BITSHIFT[crc, 8], Inline.BITSHIFT[crc, -8]];
    pointer.crcTable[i] ← crc;
    ENDLOOP;
  END;

ForceFrameTimerWakeup: ENTRY PROCEDURE = BEGIN NOTIFY frameTimer; END;

FrameTimerProcess: PROCEDURE =
  BEGIN
  ENABLE UNWIND => NULL;
  resetSynSeen: MIOCInternalD0.ResetStatusCommand ←
    [commandPart: [channelA, resetStatus, 0], dataPart: [synSeen: TRUE]];
  iocb: MIOCInternalD0.IOCBPtr;
  DO
    IF ending THEN RETURN;
    FrameTimerWait[];
    IF ending THEN RETURN;
    IF CSB.status.synSeen THEN
      [] ← MIOCInternalD0.IssueChipCommand[line, resetSynSeen]
    ELSE
      IF (iocb ← CSB.inputChain) # Zone.nil THEN
        IF ResidentHeap.first64K[iocb].start # ResidentHeap.first64K[
          iocb].offset THEN
          IF ResidentHeap.first64K[iocb].marked THEN
            BEGIN -- Should reset looking for SYNs --
            Suspend[input];
            impl.AbortTopIOCB[iocb];
            Restart[input];
            END
          ELSE ResidentHeap.first64K[iocb].marked ← TRUE;
    ENDLOOP;
  END;

FrameTimerWait: PRIVATE ENTRY PROCEDURE = BEGIN WAIT frameTimer; END;

FreeTables: PRIVATE PROCEDURE =
  BEGIN
  IF tables = NIL THEN RETURN;
```



```
  CSB.tables ← [lowHalf: NIL, highHalf: 0, highHalfPlusOne: 1];
  Heap.FreeNode[tablesZone, tables];
  SpecialHeap.MakeSwappable[tablesZone];
  Heap.Delete[tablesZone];
  tables ← NIL;
  END;

Restart: PRIVATE ENTRY PROCEDURE [queue: MIOCInternalD0.QueueType] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  SELECT queue FROM
    input => writeReg1.data.receiverInterrupts ← allWithParity;
    output => writeReg1.data.transmitterInterruptEnable ← TRUE;
    ENDCASE;
  [] ← MIOCInternalD0.IssueChipCommand[line, writeReg1];
  END;

SetCharLength: PRIVATE INTERNAL PROCEDURE [
  paramHandle: RS232CFace.ParamHandle] =
  BEGIN
  writeReg3.data.characterLength ← writeReg5.data.characterLength ←
    SELECT paramHandle.charLength FROM
      5 => lengthIs5Bits,
      6 => lengthIs6Bits,
      7 => lengthIs7Bits,
      ENDCASE --8-- => lengthIs8Bits;
  majorChange ← TRUE;
  END;

SetCorrespondent: PRIVATE INTERNAL PROCEDURE [
  paramHandle: RS232CFace.ParamHandle] =
  BEGIN
  i: CARDINAL;
  inputStart, outputStart: MIOCInternalD0.State;
  tablePtr: LONG POINTER TO MIOCInternalD0.Tables;
  inPtr: LONG POINTER TO ARRAY [20B..45B] OF ARRAY [0..17B] OF
    MIOCInternalD0.InputEvent;
  outPtr: LONG POINTER TO ARRAY [20B..45B] OF ARRAY [0..17B] OF
    MIOCInternalD0.OutputEvent;
  SELECT paramHandle.correspondent FROM
    RS232CCorrespondents.xerox800 =>
      BEGIN
      tablePtr ← AllocateTables[];
      [inputStart, outputStart] ← MIOCInternalD0.X800Init[@tablePtr.charTable];
      bit ← 100000B;
      shift1 ← 8;
      shift2 ← 1B;
      magic ← 100005B;
      rev ← TRUE;
      BuildChecksumTables[tablePtr];
      END;
    RS232CCorrespondents.xerox850 =>
      BEGIN
      tablePtr ← AllocateTables[];
      [inputStart, outputStart] ← MIOCInternalD0.X850Init[@tablePtr.charTable];
      bit ← 1B;
      shift1 ← 0B;
      shift2 ← -1B;
      magic ← 120001B;
```

```
      rev ← FALSE;
      BuildChecksumTables[tablePtr];
      END;
    RS232CCorrespondents.system6, RS232CCorrespondents.cmcII =>
      BEGIN
      tablePtr ← AllocateTables[];
      [inputStart, outputStart] ← MIOCInternalD0.System6Init[
        @tablePtr.charTable];
      bit ← 1B;
      shift1 ← 0B;
      shift2 ← -1B;
      magic ← 120001B;
      rev ← FALSE;
      BuildChecksumTables[tablePtr];
      END;
    RS232CCorrespondents.oisSystemElement => FreeTables[];
    RS232CCorrespondents.ttyHost => FreeTables[];
    255 =>
      BEGIN
      tablePtr ← AllocateTables[];
      inputStart ← 20B;
      outputStart ← 21B;
      FOR i IN [0..255] DO tablePtr.crcTable[i] ← 0; ENDLOOP;
      FOR i IN [0..255] DO tablePtr.charTable[i] ← 0; ENDLOOP;
      inPtr ← LOOPHOLE[@tablePtr.stateTables];
      inPtr[20B][0] ← [new: 20B, save: TRUE, end: TRUE];
      outPtr ← LOOPHOLE[@tablePtr.stateTables];
      outPtr[21B][0] ← [new: 21B, send: TRUE];
      END;
    ENDCASE => globalOutcome ← unimplemented;
  impl.SetStartStates[inputStart, outputStart];
  END;

SetDataTerminalReady: PRIVATE INTERNAL PROCEDURE [
  paramHandle: RS232CFace.ParamHandle] =
  BEGIN
  writeReg5.data.dtr ← paramHandle.dataTerminalReady;
  minorChange ← TRUE;
  END;

SetFrameTimeout: PRIVATE INTERNAL PROCEDURE [
  paramHandle: RS232CFace.ParamHandle] =
  BEGIN
  IF paramHandle.correspondent = RS232CCorrespondents.ttyHost AND
    paramHandle.lineType = asynchronous THEN
    BEGIN
    Process.DisableTimeout[@frameTimer];
    CSB.initialTimeoutCount ← CSB.currentTimeoutCount ←
      IF paramHandle.frameTimeout = 0 THEN MIOCInternalD0.nullTimer
      ELSE paramHandle.frameTimeout/10; -- Microcode uses 1/100 seconds --
    END
  ELSE
    BEGIN
    IF paramHandle.frameTimeout = 0 OR paramHandle.correspondent = 255 THEN
      Process.DisableTimeout[@frameTimer]
    ELSE
      BEGIN
      Process.SetTimeout[
        @frameTimer, Process.MsecToTicks[paramHandle.frameTimeout]];
      NOTIFY frameTimer;
```



```
      END;
    END;
  END;

SetLineSpeed: PRIVATE INTERNAL PROCEDURE [paramHandle: RS232CFace.ParamHandle] =
  BEGIN
  lineSpeedInfo: MIOCInternalD0.TimerSpeedConstant ←
    SELECT paramHandle.lineSpeed FROM
      bps50 => MIOCInternalD0.bps50,
      bps75 => MIOCInternalD0.bps75,
      bps110 => MIOCInternalD0.bps110,
      bps134p5 => MIOCInternalD0.bps134p5,
      bps150 => MIOCInternalD0.bps150,
      bps300 => MIOCInternalD0.bps300,
      bps600 => MIOCInternalD0.bps600,
      bps1200 => MIOCInternalD0.bps1200,
      bps2400 => MIOCInternalD0.bps2400,
      bps3600 => MIOCInternalD0.bps3600,
      bps4800 => MIOCInternalD0.bps4800,
      bps7200 => MIOCInternalD0.bps7200,
      bps9600 => MIOCInternalD0.bps9600,
      ENDCASE => MIOCInternalD0.bps9600;
  SELECT paramHandle.lineType FROM
    asynchronous =>
      MIOCInternalD0.LoadTimer[
        line, rs232cClock, squareWaveGenerator, lineSpeedInfo.x16];
    byteSynchronous =>
      MIOCInternalD0.LoadTimer[
        line, rs232cClock, squareWaveGenerator, lineSpeedInfo.x1];
    bitSynchronous =>
      BEGIN
      MIOCInternalD0.LoadTimer[
        line, rs232cClock, squareWaveGenerator, lineSpeedInfo.x1];
      CSB.currentTimeoutCount ←
        SELECT paramHandle.lineSpeed FROM
          bps50 => 24,
          bps75 => 16,
          bps110 => 11,
          bps134p5 => 9,
          bps150 => 8,
          bps300 => 4,
          bps600 => 2,
          ENDCASE => 1;
      CSB.initialTimeoutCount ← CSB.currentTimeoutCount;
      END;
    ENDCASE;
  majorChange ← TRUE;
  END;

SetLineType: PRIVATE INTERNAL PROCEDURE [paramHandle: RS232CFace.ParamHandle]
  RETURNS [BOOLEAN] =
  BEGIN
  init: MIOCInternalD0.PollerCommandData =
    [[channelA, initializeOverlay, 0], 0];
  currentMicrocode: PROGRAM ←
    SELECT paramHandle.lineType FROM
      asynchronous => MIOCInternalD0.RS232CAsyncMicrocodeD0,
      byteSynchronous => MIOCInternalD0.RS232CByteMicrocodeD0,
```

```
      autoRecognition => MIOCInternalD0.RS232CTtyMicrocodeD0,
      ENDCASE => MIOCInternalD0.RS232CBitMicrocodeD0;
    IF paramHandle.lineType = asynchronous AND paramHandle.correspondent =
      RS232CCorrespondents.ttyHost THEN
      currentMicrocode ← MIOCInternalD0.RS232CTtyMicrocodeD0;
    CSB.initialTimeoutCount ← CSB.currentTimeoutCount ← MIOCInternalD0.nullTimer;
    IF ~MIOCInternalD0.OverlayMicrocode[currentMicrocode] THEN RETURN[FALSE];
    [] ← MIOCInternalD0.IssueChipCommand[line, init];
    SELECT paramHandle.lineType FROM
      asynchronous =>
        BEGIN
        writeReg3.data.receiverCRCEnable ← FALSE;
        writeReg4.data.clockMode ← times16;
        writeReg4.data.stopBits ← two;
        writeReg5.data.transmitterCRCEnable ← FALSE;
        MIOCInternalD0.SetClocking[line, local];
        END;
      byteSynchronous =>
        BEGIN
        writeReg3.data.receiverCRCEnable ← FALSE;
        writeReg4.data.clockMode ← times1;
        writeReg4.data.stopBits ← none;
        writeReg5.data.transmitterCRCEnable ← FALSE;
        MIOCInternalD0.SetClocking[line, external];
        CSB.initialTimeoutCount ← CSB.currentTimeoutCount ← 100; -- one second --
        END;
      bitSynchronous =>
        BEGIN
        writeReg3.data.receiverCRCEnable ← TRUE;
        writeReg4.data.clockMode ← times1;
        writeReg4.data.stopBits ← none;
        writeReg4.data.syncLength ← sdlcSync;
        writeReg5.data.transmitterCRCEnable ← TRUE;
        writeReg6.syncChar ← writeReg7.syncChar ← 176B;
        MIOCInternalD0.SetClocking[line, external];
        END;
      autoRecognition =>
        BEGIN
        writeReg3.data.receiverCRCEnable ← FALSE;
        writeReg4.data.clockMode ← times1;
        writeReg4.data.stopBits ← two;
        writeReg5.data.transmitterCRCEnable ← FALSE;
        MIOCInternalD0.SetClocking[line, external];
        END;
      ENDCASE;
    majorChange ← TRUE;
    RETURN[TRUE];
    END;

  SetParity: PRIVATE INTERNAL PROCEDURE [paramHandle: RS232CFace.ParamHandle] =
    BEGIN
    SELECT paramHandle.parity FROM
      none => writeReg4.data.parity ← none;
      odd => writeReg4.data.parity ← odd;
      even => writeReg4.data.parity ← even;
      one => globalOutcome ← unimplemented;
      ENDCASE --zero-- => globalOutcome ← unimplemented;
    majorChange ← TRUE;
    END;
```



```
  SetRegistersNow: PRIVATE INTERNAL PROCEDURE [
    paramHandle: RS232CFace.ParamHandle] =
    BEGIN
    disableInterrupts: MIOCInternalD0.WriteReg1 ←
      [command: [channelA, writeChip, 1],
        data: [FALSE, FALSE, FALSE, disabled, FALSE, FALSE, FALSE]];
    resetExternal: MIOCInternalD0.WriteReg0 ←
      [command: [channelA, writeChip, 0], data: [null, resetExternalStatus, 0]];
    readReg0: MIOCInternalD0.PollerCommandData ←
      [commandPart: [channelA, readChip, 0], dataPart: 0];
    setRegister5: MIOCInternalD0.PollerCommandData ←
      [commandPart: [channelA, loadRegister5, 0], dataPart: 0];
    CSB.reg3Data ← writeReg3.data;
    CSB.reg5Data ← writeReg5.data;
    IF majorChange THEN
      BEGIN
      [] ← MIOCInternalD0.IssueChipCommand[line, disableInterrupts];
      [] ← MIOCInternalD0.IssueChipCommand[line, writeReg4];
      [] ← MIOCInternalD0.IssueChipCommand[line, writeReg6];
      [] ← MIOCInternalD0.IssueChipCommand[line, writeReg7];
      [] ← MIOCInternalD0.IssueChipCommand[line, resetExternal];
      [] ← MIOCInternalD0.IssueChipCommand[line, writeReg3];
      [] ← MIOCInternalD0.IssueChipCommand[line, setRegister5];
      [] ← MIOCInternalD0.IssueChipCommand[line, resetExternal];
      [] ← MIOCInternalD0.IssueChipCommand[line, writeReg1];
      END;
    IF minorChange THEN [] ← MIOCInternalD0.IssueChipCommand[line, setRegister5];
    majorChange ← minorChange ← FALSE;
    END;

  SetRequestToSend: PRIVATE INTERNAL PROCEDURE [
    paramHandle: RS232CFace.ParamHandle] =
    BEGIN writeReg5.data.rts ← paramHandle.requestToSend; minorChange ← TRUE; END;

  SetStopBits: PRIVATE INTERNAL PROCEDURE [paramHandle: RS232CFace.ParamHandle] =
    BEGIN
    IF paramHandle.lineType = asynchronous THEN
      BEGIN
      writeReg4.data.stopBits ←
        SELECT paramHandle.stopBits FROM 1 => one, ENDCASE --two-- => two;
      majorChange ← TRUE;
      END;
    END;

  SetSyncChar: PRIVATE INTERNAL PROCEDURE [paramHandle: RS232CFace.ParamHandle] =
    BEGIN
    IF paramHandle.lineType = byteSynchronous THEN
      BEGIN
      writeReg6.syncChar ← writeReg7.syncChar ← paramHandle.syncChar;
      majorChange ← TRUE;
      END;
    END;

  SetSyncCount: PRIVATE INTERNAL PROCEDURE [paramHandle: RS232CFace.ParamHandle] =
    BEGIN
    IF paramHandle.lineType = byteSynchronous THEN
      BEGIN
      -- writeReg4.data.syncLength ← IF paramHandle.syncCount = 1 THEN singleSync ELSE dou
**bleSync;
```

```
    writeReg4.data.syncLength ← singleSync;
    majorChange ← TRUE;
    END;
  END;

Suspend: PRIVATE ENTRY PROCEDURE [queue: MIOCInternalD0.QueueType] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  SELECT queue FROM
    input => writeReg1.data.receiverInterrupts ← disabled;
    output => writeReg1.data.transmitterInterruptEnable ← FALSE;
    ENDCASE;
  [] ← MIOCInternalD0.IssueChipCommand[line, writeReg1];
  END;
-- Main program

Process.SetTimeout[@breakTimeout, Process.MsecToTicks[250]]; --1/4 second --
END. -- MIOCCommCmdsD0
LOG
Time: September 25, 1979 4:43 PM          By: Bill Danielson      Action: Created file
Time: October 2, 1979 1:34 PM By: Bill Danielson      Action: Added routines to set various chip re
**gisters
Time: October 4, 1979 1:21 PM By: Bill Danielson Action: Added timer setting routines.
Time: November 2, 1979 6:24 PM By: Bill Danielson Action: Removed lowest level code to MIOC
**Hardware.
Time: December 3, 1979 4:19 PM By: Bill Danielson Action: Changed parameter setting code to n
**ot use channel reset.
Time: January 14, 1980 1:11 PM By: Bill Danielson Action: Changed microcode overlays back to .
**bcds.
Time: January 22, 1980 2:41 PM By: Bill Danielson Action: Added changes for RS232CEnvironme
**nt and RS232CCorrespondents.
Time: January 31, 1980 11:29 AM By: Bill Danielson Action: Added tables for X850 and catch mor
**e unimplemented parameters.
Time: March 3, 1980 12:02 PM By: Bill Danielson Action: Fix to allow setting RTS.
Time: March 5, 1980 3:52 PM By: Bill Danielson Action: Added code for tty support.
Time: March 19, 1980 1:43 PM By: Bill Danielson Action: Added code for intraframe fill.
Time: March 21, 1980 3:33 PM By: Bill Danielson Action: Allocate state tables from heap.
Time: March 24, 1980 11:34 AM By: Bill Danielson Action: Reload microcode on debugger swaps
**.
Time: April 17, 1980 5:26 PM By: Bill Danielson Action: Change CSB.tables to be microcode poin
**ter.
Time: May 22, 1980 5:16 PM By: Bill Danielson Action: Multiple line support.
Time: June 25, 1980 2:21 PM By: Bill Danielson Action: AutoRecognition.
Time: July 14, 1980 11:28 AM By: Bill Danielson Action: Try and fix microcode very dead problem
**.
Time: July 22, 1980 2:10 PM By: Bill Danielson Action: New disabling/enabling code.
Time: July 28, 1980 5:36 PM By: Bill Danielson Action: Remove RS232C heap.
Time: August 5, 1980 11:44 AM By: Bill Danielson Action: Removed Suspend/Restart from RS23
**2CFace.
Time: August 12, 1980 2:27 PM By: Bill Danielson Action: Fixed Allocate Tables.
```

```
-- File: System6TablesD0.mesa
-- LastEdited: April 9, 1980  5:45 PM   By: BRD

DIRECTORY
  Environment: FROM "Environment" USING [Byte],
  MIOCInternalD0: FROM "MIOCInternalD0" USING [
    CharTable, FillCharTable, InputEvent, OutputEvent, State, StateTable];

System6TablesD0: PROGRAM EXPORTS MIOCInternalD0 =
  BEGIN
  -- System6 Character Types Offsets
  nrm: CARDINAL = 0B;
  syn: CARDINAL = 2B;
  dle: CARDINAL = 4B;
  soh: CARDINAL = 6B;
  stx: CARDINAL = 10B;
  etb: CARDINAL = 12B;
  itb: CARDINAL = 14B;
  etx: CARDINAL = 16B;
  enq: CARDINAL = 20B;
  bel: CARDINAL = 22B;
  eot: CARDINAL = 24B;
  nak: CARDINAL = 26B;

  system6CharTable: MIOCInternalD0.CharTable =
    [
      --          0       1       2       3       4       5       6       7       8       9
**  A     B       C       D       E       F
      nrm, soh, stx, etx, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 0
      dle, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      itb, -- 1
      nrm, nrm, nrm, nrm, nrm, nrm, etb, nrm, nrm, nrm, nrm, nrm, nrm, enq, nrm,
      bel, -- 2
      nrm, nrm, syn, nrm, nrm, nrm, nrm, eot, nrm, nrm, nrm, nrm, nrm, nak, nrm,
      nrm, -- 3
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 4
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 5
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 6
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 7
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 8
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 9
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- A
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- B
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- C
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- D
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- E
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm]; -- F
  x850CharTable: MIOCInternalD0.CharTable =
    [
      --          0       1       2       3       4       5       6       7       8       9
**  A     B       C       D       E       F
      nrm, soh, stx, etx, eot, enq, nrm, bel, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 0
      dle, nrm, nrm, nrm, nrm, nak, syn, etb, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      itb, -- 1
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 2
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 3
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 4
```



```
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 5
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 6
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 7
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 8
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 9
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- A
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- B
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- C
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- D
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- E
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm]; -- F

  -- System 6 input state table
  sys6InStart: MIOCInternalD0.State = 20B;
  sys6InHeader: MIOCInternalD0.State = 21B;
  sys6InText: MIOCInternalD0.State = 22B;
  sys6InBCC1: MIOCInternalD0.State = 23B;
  sys6InBCC2: MIOCInternalD0.State = 24B;
  sys6InGotDLE: MIOCInternalD0.State = 25B;
  sys6InTrans: MIOCInternalD0.State = 26B;
  sys6InTransDLE: MIOCInternalD0.State = 27B;
  sys6InEndFrame: MIOCInternalD0.State = 30B;
  sys6InCompleted: MIOCInternalD0.State = 377B;

  System6InputTable: TYPE = ARRAY [20B..30B] OF ARRAY [0..17B] OF
    MIOCInternalD0.InputEvent;
  system6InputTable: System6InputTable =
    [[[new: sys6InStart, save: TRUE], -- sys6InStart, normal
      [new: sys6InStart, synSeen: TRUE], -- sys6InStart, syn
      [new: sys6InGotDLE, save: TRUE], -- sys6InStart, dle
      [new: sys6InHeader, save: TRUE, zero: TRUE], -- sys6InStart, soh
      [new: sys6InText, save: TRUE, zero: TRUE], -- sys6InStart, stx
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InStart, etb (E)
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InStart, itb (E)
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InStart, etx (E)
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InStart, enq
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InStart, bel
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InStart, eot
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InStart, nak
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InStart, ?
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InStart, ?
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InStart, ?
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE]], -- sys6InStart, ?
     [[new: sys6InHeader, save: TRUE, crc: TRUE], -- sys6InHeader, normal
      [new: sys6InHeader, synSeen: TRUE], -- sys6InHeader, syn
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InHeader, dle (E)
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InHeader, soh (E)
      [new: sys6InText, save: TRUE, crc: TRUE], -- sys6InHeader, stx
      [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InHeader, etb
      [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InHeader, itb
      [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InHeader, etx
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InHeader, enq (E)
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InHeader, bel (E)
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- sys6InHeader, eot (E)
      [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
```

```
    -- sys6InHeader, nak (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InHeader, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InHeader, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InHeader, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE]], -- sys6InHeader, ?
  [[new: sys6InText, save: TRUE, crc: TRUE], -- sys6InText, normal
    [new: sys6InText, synSeen: TRUE], -- sys6InText, syn
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InText, dle (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InText, soh (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InText, stx (E)
    [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InText, etb
    [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InText, itb
    [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InText, etx
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InText, enq (E)
    [new: sys6InText, save: TRUE, crc: TRUE], -- sys6InText, bel
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InText, eot (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InText, nak (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InText, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InText, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InText, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE]], -- sys6InText, ?
  [[new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, normal
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, syn
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, dle
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, soh
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, stx
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, etb
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, itb
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, etx
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, enq
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, bel
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, eot
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, nak
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, ?
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, ?
    [new: sys6InBCC2, save: TRUE, crc: TRUE], -- sys6InBCC1, ?
    [new: sys6InBCC2, save: TRUE, crc: TRUE]], -- sys6InBCC1, ?
  [[new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, normal
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, syn
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, dle
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, soh
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, stx
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, etb
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, itb
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, etx
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, enq
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, bel
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, eot
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, nak
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, ?
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, ?
    [new: sys6InEndFrame, save: TRUE, crc: TRUE], -- sys6InBCC2, ?
    [new: sys6InEndFrame, save: TRUE, crc: TRUE]], -- sys6InBCC2, ?
  [[new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, normal
    [new: sys6InGotDLE, synSeen: TRUE], -- sys6InGotDLE, syn
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, dle
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, soh
    [new: sys6InTrans, save: TRUE, zero: TRUE], -- sys6InGotDLE, stx
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, etb
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, itb
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, etx
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
```



```
    -- sys6InGotDLE, enq
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, bel
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, eot
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InGotDLE, nak
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InGotDLE, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InGotDLE, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE], -- sys6InGotDLE, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE]], -- sys6InGotDLE, ?
  [[new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, normal
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, syn
    [new: sys6InTransDLE, save: TRUE], -- sys6InTrans, dle
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, soh
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, stx
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, etb
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, itb
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, etx
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, enq
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, bel
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, eot
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, nak
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, ?
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, ?
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTrans, ?
    [new: sys6InTrans, save: TRUE, crc: TRUE]], -- sys6InTrans, ?
  [[new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, normal (E)
    [new: sys6InTrans, synSeen: TRUE, back: TRUE], -- sys6InTransDLE, syn
    [new: sys6InTrans, save: TRUE, crc: TRUE], -- sys6InTransDLE, dle
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, soh (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, stx (E)
    [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InTransDLE, etb
    [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InTransDLE, itb
    [new: sys6InBCC1, save: TRUE, crc: TRUE], -- sys6InTransDLE, etx
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, enq (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, bel (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, eot (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, nak (E)
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- sys6InTransDLE, ?
    [new: sys6InStart, save: TRUE, zero: TRUE, end: TRUE]],
  -- sys6InTransDLE, ?
  [[new: sys6InStart, end: TRUE], -- sys6InEndFrame, normal
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, syn
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, dle
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, soh
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, stx
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, etb
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, itb
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, etx
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, enq
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, bel
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, eot
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, nak
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, ?
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, ?
    [new: sys6InStart, end: TRUE], -- sys6InEndFrame, ?
    [new: sys6InStart, end: TRUE]]]; -- sys6InEndFrame, ?


-- System 6 output state table
sys6OutStart: MIOCInternalDO.State = 31B;
```

```
  sys6OutHeader: MIOCInternalDO.State = 32B;
  sys6OutText: MIOCInternalDO.State = 33B;
  sys6OutBCC1: MIOCInternalDO.State = 34B;
  sys6OutBCC2: MIOCInternalDO.State = 35B;
  sys6OutGotDLE: MIOCInternalDO.State = 36B;
  sys6OutTrans: MIOCInternalDO.State = 37B;
  sys6OutTransDLE: MIOCInternalDO.State = 40B;
  sys6OutCompleted: MIOCInternalDO.State = 377B;
  System6OutputTable: TYPE = ARRAY [31B..40B] OF ARRAY [0..17B] OF
    MIOCInternalDO.OutputEvent;
  system6OutputTable: System6OutputTable =
    [[[new: sys6OutStart, send: TRUE], -- sys6OutStart, normal
       [new: sys6OutStart, synSeen: TRUE, send: TRUE], -- sys6OutStart, syn
       [new: sys6OutGotDLE, send: TRUE], -- sys6OutStart, dle
       [new: sys6OutHeader, send: TRUE, zero: TRUE], -- sys6OutStart, soh
       [new: sys6OutText, send: TRUE, zero: TRUE], -- sys6OutStart, stx
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutStart, etb (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutStart, itb (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutStart, etx (E)
       [new: sys6OutStart, send: TRUE], -- sys6OutStart, enq
       [new: sys6OutStart, send: TRUE], -- sys6OutStart, bel
       [new: sys6OutStart, send: TRUE], -- sys6OutStart, eot
       [new: sys6OutStart, send: TRUE], -- sys6OutStart, nak
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutStart, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutStart, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutStart, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE]], -- sys6OutStart, ?
     [[new: sys6OutHeader, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutHeader, normal
       [new: sys6OutHeader, synSeen: TRUE, send: TRUE, fillOK: TRUE],
       -- sys6OutHeader, syn
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, dle (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, soh (E)
       [new: sys6OutText, send: TRUE, crc: TRUE], -- sys6OutHeader, stx
       [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
       -- sys6OutHeader, etb
       [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
       -- sys6OutHeader, itb
       [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
       -- sys6OutHeader, etx
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, enq (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, bel (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, eot (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, nak (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutHeader, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE]], -- sys6OutHeader, ?
     [[new: sys6OutText, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutText, normal
       [new: sys6OutText, synSeen: TRUE, send: TRUE, fillOK: TRUE],
       -- sys6OutText, syn
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, dle (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, soh (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, stx (E)
       [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE], -- sys6OutText, etb
       [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE], -- sys6OutText, itb
       [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE], -- sys6OutText, etx
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, enq (E)
       [new: sys6OutText, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutText, bel
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, eot (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, nak (E)
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutText, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE]], -- sys6OutText, ?
     [[new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, normal
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, syn
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, dle
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, soh
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, stx
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, etb
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, itb
```



```
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, etx
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, enq
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, bel
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, eot
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, nak
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, ?
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, ?
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE], -- sys6OutBCC1, ?
       [new: sys6OutBCC2, back: TRUE, sendCRC: TRUE]], -- sys6OutBCC1, ?
     [[new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, normal
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, syn
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, dle
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, soh
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, stx
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, etb
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, itb
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, etx
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, enq
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, bel
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, eot
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, nak
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, ?
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, ?
       [new: sys6OutStart, sendCRC: TRUE], -- sys6OutBCC2, ?
       [new: sys6OutStart, sendCRC: TRUE]], -- sys6OutBCC2, ?
     [[new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, normal
       [new: sys6OutStart, synSeen: TRUE, send: TRUE, zero: TRUE],
       -- sys6OutGotDLE, syn
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, dle
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, soh
       [new: sys6OutTrans, send: TRUE, zero: TRUE], -- sys6OutGotDLE, stx
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, etb
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, itb
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, etx
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, enq
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, bel
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, eot
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, nak
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutGotDLE, ?
       [new: sys6OutStart, send: TRUE, zero: TRUE]], -- sys6OutGotDLE, ?
     [[new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, normal
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, syn
       [new: sys6OutTransDLE, send: TRUE], -- sys6OutTrans, dle
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, soh
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, stx
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, etb
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, itb
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, etx
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, enq
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, bel
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, eot
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, nak
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, ?
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, ?
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE],
       -- sys6OutTrans, ?
       [new: sys6OutTrans, send: TRUE, crc: TRUE, fillOK: TRUE]],
     -- sys6OutTrans, ?
     [[new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, normal (E)
```

```
      [new: sys6OutTrans, synSeen: TRUE, send: TRUE], -- sys6OutTransDLE, syn
      [new: sys6OutTrans, send: TRUE, crc: TRUE], -- sys6OutTransDLE, dle
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, soh (E)
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, stx (E)
      [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
      -- sys6OutTransDLE, etb
      [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
      -- sys6OutTransDLE, itb
      [new: sys6OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
      -- sys6OutTransDLE, etx
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, enq (E)
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, bel (E)
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, eot (E)
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, nak (E)
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, ?
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, ?
      [new: sys6OutStart, send: TRUE, zero: TRUE], -- sys6OutTransDLE, ?
      [new: sys6OutStart, send: TRUE, zero: TRUE]]; -- sys6OutTransDLE, ?
-- Fill Character Tables
asciiSYN: Environment.Byte = 26B;
asciiDLE: Environment.Byte = 20B;
asciiFillCharTable: MIOCInternalD0.FillCharTable =
  [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
    -- 20B - 30B
    [asciiSYN, asciiSYN], -- sys6OutStart
    [asciiSYN, asciiSYN], -- sys6OutHeader
    [asciiSYN, asciiSYN], -- sys6OutText
    [asciiSYN, asciiSYN], -- sys6OutBCC1
    [asciiSYN, asciiSYN], -- sys6OutBCC2
    [asciiSYN, asciiSYN], -- sys6OutGotDLE
    [asciiDLE, asciiSYN], -- sys6OutTrans
    [asciiDLE, asciiSYN], -- sys6OutTransDLE
    [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]; -- 41B..45B
ebcdicSYN: Environment.Byte = 62B;
ebcdicDLE: Environment.Byte = 20B;
ebcdicFillCharTable: MIOCInternalD0.FillCharTable =
  [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
    -- 20B - 30B
    [ebcdicSYN, ebcdicSYN], -- sys6OutStart
    [ebcdicSYN, ebcdicSYN], -- sys6OutHeader
    [ebcdicSYN, ebcdicSYN], -- sys6OutText
    [ebcdicSYN, ebcdicSYN], -- sys6OutBCC1
    [ebcdicSYN, ebcdicSYN], -- sys6OutBCC2
    [ebcdicSYN, ebcdicSYN], -- sys6OutGotDLE
    [ebcdicDLE, ebcdicSYN], -- sys6OutTrans
    [ebcdicDLE, ebcdicSYN], -- sys6OutTransDLE
    [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]; -- 41B..45B
System6Init: PUBLIC PROCEDURE [pointer: LONG POINTER]
  RETURNS [inputStart, outputStart: MIOCInternalD0.State] =
  BEGIN
  charPtr: LONG ORDERED POINTER TO MIOCInternalD0.CharTable;
  inputPtr: LONG ORDERED POINTER TO System6InputTable;
  outputPtr: LONG ORDERED POINTER TO System6OutputTable;
  fillPtr: LONG ORDERED POINTER TO MIOCInternalD0.FillCharTable;
  charPtr ← LOOPHOLE[pointer];
  charPtr↑ ← system6CharTable;
  inputPtr ← LOOPHOLE[charPtr + SIZE[MIOCInternalD0.CharTable]];
  inputPtr↑ ← system6InputTable;
  outputPtr ← LOOPHOLE[inputPtr + SIZE[System6InputTable]];
  outputPtr↑ ← system6OutputTable;
  fillPtr ← LOOPHOLE[inputPtr + SIZE[MIOCInternalD0.StateTable]];
  fillPtr↑ ← ebcdicFillCharTable;
  RETURN[sys6InStart, sys6OutStart];
  END;

X850Init: PUBLIC PROCEDURE [pointer: LONG POINTER]
  RETURNS [inputStart, outputStart: MIOCInternalD0.State] =
  BEGIN
  charPtr: LONG ORDERED POINTER TO MIOCInternalD0.CharTable;
  inputPtr: LONG ORDERED POINTER TO System6InputTable;
  outputPtr: LONG ORDERED POINTER TO System6OutputTable;
  fillPtr: LONG ORDERED POINTER TO MIOCInternalD0.FillCharTable;
  charPtr ← LOOPHOLE[pointer];
  charPtr↑ ← x850CharTable;
```



```
  inputPtr ← LOOPHOLE[charPtr + SIZE[MIOCInternalD0.CharTable]];
  inputPtr↑ ← system6InputTable;
  outputPtr ← LOOPHOLE[inputPtr + SIZE[System6InputTable]];
  outputPtr↑ ← system6OutputTable;
  fillPtr ← LOOPHOLE[inputPtr + SIZE[MIOCInternalD0.StateTable]];
  fillPtr↑ ← asciiFillCharTable;
  RETURN[sys6InStart, sys6OutStart];
  END;

END.  -- System6TablesD0
LOG
Time: August 30, 1979  2:38 PM By: BIll Danielson Action: Created file
Time: September 19, 1979  3:11 PM By: BIll Danielson Action: Combined common tables and added
fill character tables.
Time: October 4, 1979  1:11 PM By: BIll Danielson Action: Modified for D0 implementation.
Time: November 29, 1979  2:15 PM By: BIll Danielson Action: Broke off to just include System
6 tables.
Time: January 31, 1980  11:17 AM By: BIll Danielson Action: Added ascii character table for
Xerox850.
Time: February 13, 1980  2:50 PM By: BIll Danielson Action: Fixed stop code (BEL) states.
Time: March 19, 1980  10:54 AM By: Bill Danielson Action: Added intraframe fill characters.
Time: April 9, 1980  5:45 PM By: Bill Danielson Action: Reduce global frame usage.
```

```
-- TTYPortHeadD0.mesa
-- Last edited: October 7, 1980 7:04 PM By: Mary Artibee

DIRECTORY
  TTYPortFace: FROM "TTYPortFace" USING [
    CharacterLength, DeviceStatus, LineSpeed, Parameter, Parity, StopBits,
    TransferStatus],
  MIOCInternalD0: FROM "MIOCInternalD0" USING [
    bps50, bps75, bps110, bps134p5, bps150, bps300, bps600, bps1200, bps1800,
    bps2000, bps2400, bps3600, bps4800, bps7200, bps9600, bps19200, GetCount,
    InitializePrinter, LoadTimer, LoadTTYPortRegister, maxLines,
    TimerSpeedConstant, TTYPortLoadWord, TTYPortStatusWord, UsartStatus];

TTYPortHeadD0: PROGRAM IMPORTS MIOCInternalD0 EXPORTS TTYPortFace =
  BEGIN
  -- various definitions
  lineState: ARRAY [0..MIOCInternalD0.maxLines) OF LineRecord ← ALL[
    defaultLineRecord];
  LineRecord: TYPE = RECORD [
    usartMode: UsartModeInstr,
    usartStat: MIOCInternalD0.UsartStatus,
    deviceStatus: TTYPortFace.DeviceStatus,
    clearToSend: BOOLEAN,
    dataSetReady: BOOLEAN,
    lineSpeed: TTYPortFace.LineSpeed];
  defaultLineRecord: LineRecord =
    [usartMode: [two, none, lengthIs8bits, times16],
      usartStat: [FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE],
      deviceStatus: [FALSE, FALSE, FALSE, FALSE], clearToSend: FALSE,
      dataSetReady: FALSE, lineSpeed: bps1200];
  -- various definitions found in MIOCInternalD0
  -- TTYPortLoadWord: TYPE = MACHINE DEPENDENT RECORD [to send cmd/data to 8251
  -- unused: [0..17B],
  -- ptA0CTSToTTYPort: BOOLEAN, this is PTClearToSend to device; reflects CTS in lineRecord
  -- cmdSelect: {unused0, writeTTYPort, readTTYPort, unused3, unused4, writeUsart, readUsart,
**unused7},
  -- dataBits: [0..377B] ← 0];
  -- TTYPortStatusWord: TYPE = MACHINE DEPENDENT RECORD.[to receive status/data from 8
**251
  -- busy: BOOLEAN, this is PTCS (chip select)
  -- ptA0: BOOLEAN, this is PTClearToSend to device
  -- ptA1: BOOLEAN, this is Command/Data' to Usart
  -- notPtRD: BOOLEAN,
  -- notPtWR: BOOLEAN,
  -- notPtRTSFromTTYPort: BOOLEAN, RTS comes in on pin 16 on 16-pin dip, pin 4 on 25-pin; this
**with TxEnable allows transmission to device, ON means device powered on
  -- notPtCTSToTTYPort: BOOLEAN, this is just ptA0' to device
  -- notPtDTRFromTTYPort: BOOLEAN, DTR comes in on pin 14 on 16-pin dip, pin 20 on 25-pin, de
**vice should hold this ON always
  -- dataBits: SELECT OVERLAID * FROM
  -- readTTYPort => [char: CHARACTER],
  -- readUsart => [stat: UsartStatus],
  -- ENDCASE];
  -- UsartStatus: TYPE = MACHINE DEPENDENT RECORD [
  -- TTYPortStatusWord.dataBits on readUsart
  -- dsr: BOOLEAN, on if DSR' is 0, on MIOC means something is plugged in
  -- breakDetect: BOOLEAN,
  -- framingError: BOOLEAN,
```



```
  -- overrunError: BOOLEAN,
  -- parityError: BOOLEAN,
  -- txmtEmpty: BOOLEAN, output buffer empty
  -- rcvReady: BOOLEAN, input buffer full (masked/held Reset by RxEN off) = Wakeup
  -- txmtRegReady: BOOLEAN output buffer empty (TxRdy pin = Wakeup = empty + TxEN + CTS' l
**ow) ];
  UsartCmdType: TYPE = {
    modeSet, internalReset, errorReset, disableRxTx, enableRxTx};
  UsartModeInstr: TYPE = MACHINE DEPENDENT RECORD [
    -- MIOCInternalD0.TTYPortLoadWord.dataBits on writeUsart following RESET, UsartCmdType
** = modeSet
    stopBits: TTYPortFace.StopBits,
    parityBits: ParityBits,
    characterLength: TTYPortFace.CharacterLength,
    baudRateFactor: {syncMode, times1, times16, times64}];
  ParityBits: TYPE = {none, unused, odd, even};
  ConvertParityToParityBits: ARRAY TTYPortFace.Parity OF ParityBits =
    [none, odd, even];
  UsartCmdInstr: TYPE = MACHINE DEPENDENT RECORD [
    -- MIOCInternalD0.TTYPortLoadWord.dataBits on writeUsart other than following RESET
    enterHuntMode: BOOLEAN,
    internalReset: BOOLEAN, -- set on UsartCmdType = internalReset
    setRTS: BOOLEAN, -- not used on MIOC
    errorReset: BOOLEAN, -- set on UsartCmdType = errorReset
    sendBreak: BOOLEAN,
    rcvEnable: BOOLEAN,
    -- enables RxRdy to signal wakeups for char rcvd from device
    setDTR: BOOLEAN, -- not used on MIOC
    txmtEnable: BOOLEAN
    -- with CTS' from device (PTReqToSend pin 1 on 16-pin dip, pin 4 on 25-pin), enables transmis
**sion to device --
    ];
  -- Interface Procedures, alphabetically
  GetCommand: PUBLIC PROCEDURE [lineNumber: CARDINAL]
    RETURNS [data: CHARACTER, stat: TTYPortFace.TransferStatus] =
    BEGIN
    c: CHARACTER;
    ttyPortLoadWord: MIOCInternalD0.TTYPortLoadWord ←
      [0, lineState[lineNumber].clearToSend, readTTYPort, 0];
    ttyPortStatusWord: MIOCInternalD0.TTYPortStatusWord;
    UpdateStatus[lineNumber];
    IF ~lineState[lineNumber].deviceStatus.readyToGet THEN RETURN[c, notReady];
    ttyPortStatusWord ← MIOCInternalD0.LoadTTYPortRegister[
      lineNumber, ttyPortLoadWord];
    c ← ttyPortStatusWord.char;
    ttyPortLoadWord ← [0, lineState[lineNumber].clearToSend, readUsart, 0];
    ttyPortStatusWord ← MIOCInternalD0.LoadTTYPortRegister[
      lineNumber, ttyPortLoadWord];
    RETURN[c, UpdateTransferStatus[lineNumber, ttyPortStatusWord]];
    END;

  GetLineCount: PUBLIC PROCEDURE RETURNS [lineCount: CARDINAL] =
    BEGIN RETURN[MIOCInternalD0.GetCount[]] END;

  GetStatus: PUBLIC PROCEDURE [lineNumber: CARDINAL]
    RETURNS [stat: TTYPortFace.DeviceStatus] =
    BEGIN
    UpdateStatus[lineNumber];
```

```
    RETURN[lineState[lineNumber].deviceStatus];
    END;

  Off: PUBLIC PROCEDURE [lineNumber: CARDINAL] =
    BEGIN
    IssueUsartCmd[lineNumber, disableRxTx]; -- disables Rx and Tx
    MIOCInternalD0.InitializePrinter[lineNumber, 0];
    -- zeros mask in CSB.printerWakeup

    END;

  On: PUBLIC PROCEDURE [lineNumber: CARDINAL, mask: UNSPECIFIED] =
    BEGIN
    -- InitializePrinter, CSB.printerWakeup, LoadTimer.printerClock reference the TTYPort
    MIOCInternalD0.InitializePrinter[lineNumber, mask];
    -- sticks mask in CSB.printerWakeup
    lineState[lineNumber] ← defaultLineRecord;
    SetLineSpeed[lineNumber]; -- sets BRG printerClock line speed (1200, x16)
    IssueHardReset[lineNumber]; -- puts 8251 into known reset state
    IssueUsartCmd[lineNumber, modeSet];
    -- sets 8251 mode (2 stopbits, no parity, 7 databits, x16 clk)
    IssueUsartCmd[lineNumber, enableRxTx]; -- enables Rx and Tx

    END;

  PutCommand: PUBLIC PROCEDURE [lineNumber: CARDINAL, data: CHARACTER]
    RETURNS [stat: TTYPortFace.TransferStatus] =
    BEGIN
    ttyPortLoadWord: MIOCInternalD0.TTYPortLoadWord ←
      [0, lineState[lineNumber].clearToSend, writeTTYPort, 0];
    ttyPortStatusWord: MIOCInternalD0.TTYPortStatusWord;
    ttyPortLoadWord.dataBits ← LOOPHOLE[data];
    UpdateStatus[lineNumber];
    IF ~lineState[lineNumber].deviceStatus.readyToPut THEN RETURN[notReady];
    ttyPortStatusWord ← MIOCInternalD0.LoadTTYPortRegister[
      lineNumber, ttyPortLoadWord];
    RETURN[UpdateTransferStatus[lineNumber, ttyPortStatusWord]];
    END;

  SetParameter: PUBLIC PROCEDURE [
    lineNumber: CARDINAL, parameter: TTYPortFace.Parameter] =
    BEGIN
    WITH parameter SELECT FROM
      characterLength =>
        BEGIN
        IF lineState[lineNumber].usartMode.characterLength # characterLength THEN
          BEGIN
          lineState[lineNumber].usartMode.characterLength ← characterLength;
          IssueUsartCmd[lineNumber, internalReset];
          -- issue internalReset to prepare for mode instr
          IssueUsartCmd[lineNumber, modeSet];
          IssueUsartCmd[lineNumber, enableRxTx]; -- issue vanilla cmd instr

          END;
        END;
      clearToSend =>
        -- on the MIOC, this is PTA0; PTClearToSend on pin2 of the 16-pin dip, pin 5 on 25-pin
        BEGIN
        IF lineState[lineNumber].clearToSend # clearToSend THEN
```



```
          BEGIN
          lineState[lineNumber].clearToSend ← clearToSend;
          IssueUsartCmd[lineNumber, enableRxTx];
          END;
        END;
      dataSetReady =>
        -- on the MIOC, this is the LSIReset line; PTDataSetReady pin3 on 16-pin dip, pin 6 on 25-pi
**n
        BEGIN
        IF lineState[lineNumber].dataSetReady # dataSetReady THEN
          BEGIN
          lineState[lineNumber].dataSetReady ← dataSetReady;
          IssueUsartCmd[lineNumber, enableRxTx];
          END;
        END;
      lineSpeed =>
        BEGIN
        IF lineState[lineNumber].lineSpeed # lineSpeed THEN
          BEGIN
          lineState[lineNumber].lineSpeed ← lineSpeed;
          SetLineSpeed[lineNumber];
          END;
        END;
      parity =>
        BEGIN
        IF lineState[lineNumber].usartMode.parityBits # ConvertParityToParityBits[
          parity] THEN
          BEGIN
          lineState[lineNumber].usartMode.parityBits ← ConvertParityToParityBits[
            parity];
          IssueUsartCmd[lineNumber, internalReset];
          -- issue internalReset to prepare for mode instr
          IssueUsartCmd[lineNumber, modeSet];
          IssueUsartCmd[lineNumber, enableRxTx]; -- issue vanilla cmd instr

          END;
        END;
      stopBits =>
        BEGIN
        IF lineState[lineNumber].usartMode.stopBits # stopBits THEN
          BEGIN
          lineState[lineNumber].usartMode.stopBits ← stopBits;
          IssueUsartCmd[lineNumber, internalReset];
          -- issue internalReset to prepare for mode instr
          IssueUsartCmd[lineNumber, modeSet];
          IssueUsartCmd[lineNumber, enableRxTx]; -- issue vanilla cmd instr

          END;
        END;
      ENDCASE;
    END;
  -- Internal Procedures, alphabetically

  UpdateStatus: PRIVATE PROCEDURE [lineNumber: CARDINAL] =
    BEGIN
    ttyPortStatusWord: MIOCInternalD0.TTYPortStatusWord;
    ttyPortLoadWord: MIOCInternalD0.TTYPortLoadWord ←
      [0, lineState[lineNumber].clearToSend, readUsart, 0];
    ttyPortStatusWord ← MIOCInternalD0.LoadTTYPortRegister[
      lineNumber, ttyPortLoadWord];
```

```
    lineState[lineNumber].usartStat ← ttyPortStatusWord.stat;
    lineState[lineNumber].deviceStatus.dataTerminalReady ←
      ~ttyPortStatusWord.notPtDTRFromTTYPort;
    lineState[lineNumber].deviceStatus.readyToGet ← lineState[
      lineNumber].usartStat.rcvReady;
    lineState[lineNumber].deviceStatus.readyToPut ← lineState[
      lineNumber].usartStat.txmtRegReady;
    lineState[lineNumber].deviceStatus.requestToSend ←
      ~ttyPortStatusWord.notPtRTSFromTTYPort;
    END;

  UpdateTransferStatus: PRIVATE PROCEDURE [
    lineNumber: CARDINAL, currentStat: MIOCInternalD0.TTYPortStatusWord]
    RETURNS [transferStatus: TTYPortFace.TransferStatus] =
    BEGIN
    transferStat: TTYPortFace.TransferStatus ← success;
    lineState[lineNumber].usartStat ← currentStat.stat;
    lineState[lineNumber].deviceStatus.dataTerminalReady ←
      ~currentStat.notPtDTRFromTTYPort;
    lineState[lineNumber].deviceStatus.readyToGet ← currentStat.stat.rcvReady;
    lineState[lineNumber].deviceStatus.readyToPut ← currentStat.stat.txmtEmpty;
    lineState[lineNumber].deviceStatus.requestToSend ←
      ~currentStat.notPtRTSFromTTYPort;
    IF lineState[lineNumber].usartStat.framingError THEN
      transferStat ← asynchFramingError;
    IF lineState[lineNumber].usartStat.overrunError THEN transferStat ← dataLost;
    IF lineState[lineNumber].usartStat.parityError THEN
      transferStat ← parityError;
    IF lineState[lineNumber].usartStat.breakDetect THEN
      transferStat ← breakDetected;
    IF transferStat # success THEN
      BEGIN IssueUsartCmd[lineNumber, errorReset]; UpdateStatus[lineNumber]; END;
    RETURN[transferStat];
    END;

  IssueUsartCmd: PRIVATE PROCEDURE [lineNumber: CARDINAL, cmd: UsartCmdType] =
    BEGIN
    usartWord: UsartCmdInstr ←
      [FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, TRUE];
    ttyPortLoadWord: MIOCInternalD0.TTYPortLoadWord ←
      [0, lineState[lineNumber].clearToSend, writeUsart, 0];
    SELECT cmd FROM
      modeSet => BEGIN usartWord ← LOOPHOLE[lineState[lineNumber].usartMode]; END;
      internalReset => BEGIN usartWord.internalReset ← TRUE; END;
      errorReset => BEGIN usartWord.errorReset ← TRUE; END;
      disableRxTx =>
        BEGIN usartWord.rcvEnable ← FALSE; usartWord.txmtEnable ← FALSE; END;
      ENDCASE;
    ttyPortLoadWord.dataBits ← LOOPHOLE[usartWord];
    [] ← MIOCInternalD0.LoadTTYPortRegister[lineNumber, ttyPortLoadWord];
    END;

  SetLineSpeed: PRIVATE PROCEDURE [lineNumber: CARDINAL] =
    BEGIN
    -- issue LoadTimer to set line speed;
    lineSpeedInfo: MIOCInternalD0.TimerSpeedConstant ←
      SELECT lineState[lineNumber].lineSpeed FROM
        bps50 => MIOCInternalD0.bps50,
```



```
        bps75 => MIOCInternalD0.bps75,
        bps110 => MIOCInternalD0.bps110,
        bps134p5 => MIOCInternalD0.bps134p5,
        bps150 => MIOCInternalD0.bps150,
        bps300 => MIOCInternalD0.bps300,
        bps600 => MIOCInternalD0.bps600,
        bps1200 => MIOCInternalD0.bps1200,
        bps1800 => MIOCInternalD0.bps1800,
        bps2000 => MIOCInternalD0.bps2000,
        bps2400 => MIOCInternalD0.bps2400,
        bps3600 => MIOCInternalD0.bps3600,
        bps4800 => MIOCInternalD0.bps4800,
        bps7200 => MIOCInternalD0.bps7200,
        bps9600 => MIOCInternalD0.bps9600,
        ENDCASE => MIOCInternalD0.bps19200;
    MIOCInternalD0.LoadTimer[
      lineNumber, printerClock, squareWaveGenerator, lineSpeedInfo.x16]
    END;

  IssueHardReset: PRIVATE PROCEDURE [lineNumber: CARDINAL] =
    BEGIN
    resetUsart0: UsartCmdInstr =
      [TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE];
    resetUsart1: UsartCmdInstr =
      [FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE];
    resetUsart2: UsartCmdInstr =
      [FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE];
    ttyPortLoadWord: MIOCInternalD0.TTYPortLoadWord ←
      [0, lineState[lineNumber].clearToSend, writeUsart, 0];
    ttyPortLoadWord.dataBits ← LOOPHOLE[resetUsart0];
    [] ← MIOCInternalD0.LoadTTYPortRegister[lineNumber, ttyPortLoadWord];
    ttyPortLoadWord.dataBits ← LOOPHOLE[resetUsart1];
    [] ← MIOCInternalD0.LoadTTYPortRegister[lineNumber, ttyPortLoadWord];
    ttyPortLoadWord.dataBits ← LOOPHOLE[resetUsart2];
    [] ← MIOCInternalD0.LoadTTYPortRegister[lineNumber, ttyPortLoadWord];
    END;

  END. -- TTYPortHeadD0
```

LOG

Time: July 18, 1980 4:04 PM By: Mary Artibee Action: Created file from FrontHeadD0.
Time: July 21, 1980 4:11 PM By: Mary Artibee Action: Many changes.
Time: July 22, 1980 7:41 PM By: Mary Artibee Action: LoadTTYPortRegister, TTYPortLoad
***Word, TTYPortStatusWord, UsartStatus moved to MIOCInternalD0.*
Time: July 23, 1980 4:27 PM By: Mary Artibee Action: Use LOOPHOLE instead of Inline.BIT
***OR.*
Time: July 25, 1980 2:14 PM By: Mary Artibee Action: Changed CTS, DSR defaults to FALS
***E.*
Time: August 4, 1980 5:39 PM By: Mary Artibee Action: Removed RTS test in PutCommand;
***default success in UpdateTransferStatus.*
Time: August 6, 1980 3:42 PM By: Mary Artibee Action: characterLength default now lengthI
***s8bits.*
Time: September 19, 1980 1:52 PM By: Mary Artibee Action: Do BreakDetect after Fra
***mingError.*
Time: October 7, 1980 7:04 PM By: Mary Artibee Action: Use txmtRegReady for readyToPut.

```
-- File: X800TablesD0.mesa
-- LastEdited: April 9, 1980 6:11 PM By: BRD

DIRECTORY
  Environment: FROM "Environment" USING [Byte],
  MIOCInternalD0: FROM "MIOCInternalD0" USING [
    CharTable, InputEvent, OutputEvent, State];

X800TablesD0: PROGRAM EXPORTS MIOCInternalD0 =
  BEGIN
  -- X800 Character Types Offsets
  nrm: CARDINAL = 0B;
  dle: CARDINAL = 2B;
  soh: CARDINAL = 4B;
  stx: CARDINAL = 6B;
  etb: CARDINAL = 10B;
  itb: CARDINAL = 12B;
  etx: CARDINAL = 14B;
  enq: CARDINAL = 16B;
  bel: CARDINAL = 20B;
  eot: CARDINAL = 22B;
  nak: CARDINAL = 24B;

  x800CharTable: MIOCInternalD0.CharTable =
    [
      --    0       1       2       3       4       5       6       7
**8       9       A       B       C       D       E       F
      nrm, soh, stx, etx, eot, enq, nrm, bel, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 0
      dle, nrm, nrm, nrm, nrm, nak, nrm, etb, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      itb, -- 1
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 2
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 3
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 4
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 5
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 6
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 7
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 8
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- 9
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- A
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- B
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- C
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- D
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
      nrm, -- E
      nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm, nrm,
```



```
      nrm]; -- F

  -- Xerox 800 input state table
  x800InStart: MIOCInternalD0.State = 20B;
  x800InHeader: MIOCInternalD0.State = 21B;
  x800InText: MIOCInternalD0.State = 22B;
  x800InBCC1: MIOCInternalD0.State = 23B;
  x800InBCC2: MIOCInternalD0.State = 24B;
  x800InGotDLE: MIOCInternalD0.State = 25B;
  x800InCompleted: MIOCInternalD0.State = 377B;

  X800InputTable: TYPE = ARRAY [20B..25B] OF ARRAY [0..17B] OF
    MIOCInternalD0.InputEvent;
  x800InputTable: X800InputTable =
    [[[new: x800InStart, save: TRUE], -- x800InStart, normal
      [new: x800InGotDLE, save: TRUE], -- x800InStart, dle
      [new: x800InHeader, save: TRUE, zero: TRUE], -- x800InStart, soh
      [new: x800InText, save: TRUE, zero: TRUE], -- x800InStart, stx
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InStart, etb (E)
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InStart, itb (E)
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InStart, etx (E)
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, enq
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, bel
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, eot
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, nak
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InStart, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE]], -- x800InStart, ?
     [[new: x800InHeader, save: TRUE, crc: TRUE], -- x800InHeader, normal
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InHeader, dle (E)
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InHeader, soh (E)
      [new: x800InText, save: TRUE, crc: TRUE], -- x800InHeader, stx
      [new: x800InBCC1, save: TRUE, crc: TRUE], -- x800InHeader, etb
      [new: x800InBCC1, save: TRUE, crc: TRUE], -- x800InHeader, itb
      [new: x800InBCC1, save: TRUE, crc: TRUE], -- x800InHeader, etx
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InHeader, enq (E)
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InHeader, bel
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InHeader, eot (E)
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InHeader, nak (E)
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InHeader, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InHeader, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InHeader, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InHeader, ?
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE]], -- x800InHeader, ?
     [[new: x800InText, save: TRUE, crc: TRUE], -- x800InText, normal
      [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
      -- x800InText, dle (E)
```

```
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InText, soh (E)
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InText, stx (E)
    [new: x800InBCC1, save: TRUE, crc: TRUE], -- x800InText, etb
    [new: x800InBCC1, save: TRUE, crc: TRUE], -- x800InText, itb
    [new: x800InBCC1, save: TRUE, crc: TRUE], -- x800InText, etx
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InText, enq (E)
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InText, bel
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InText, eot (E)
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InText, nak (E)
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InText, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InText, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InText, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InText, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE]], -- x800InText, ?
  [[new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, normal
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, dle
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, soh
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, stx
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, etb
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, itb
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, etx
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, enq
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, bel
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, eot
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, nak
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, ?
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, ?
    [new: x800InBCC2, save: TRUE, crc: TRUE], -- x800InBCC1, ?
    [new: x800InBCC2, save: TRUE, crc: TRUE]], -- x800InBCC1, ?
  [[new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, normal
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, dle
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, soh
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, stx
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, etb
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, itb
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, etx
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, enq
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, bel
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, eot
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, nak
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, ?
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, ?
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE], -- x800InBCC2, ?
    [new: x800InStart, save: TRUE, crc: TRUE, end: TRUE]], -- x800InBCC2, ?
  [[new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, normal
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, dle
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, soh
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
```



```
    -- x800InGotDLE, stx
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, etb
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, itb
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, etx
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, enq
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, bel
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, eot
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE],
    -- x800InGotDLE, nak
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InGotDLE, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InGotDLE, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InGotDLE, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE], -- x800InGotDLE, ?
    [new: x800InStart, save: TRUE, zero: TRUE, end: TRUE]]];
  -- x800InTransDLE, ?


-- Xerox 800 output state table
x800OutStart: MIOCInternalD0.State = 26B;
x800OutHeader: MIOCInternalD0.State = 27B;
x800OutText: MIOCInternalD0.State = 30B;
x800OutBCC1: MIOCInternalD0.State = 31B;
x800OutBCC2: MIOCInternalD0.State = 32B;
x800OutGotDLE: MIOCInternalD0.State = 33B;
x800OutCompleted: MIOCInternalD0.State = 377B;
X800OutputTable: TYPE = ARRAY [26B..33B] OF ARRAY [0..17B] OF
  MIOCInternalD0.OutputEvent;
x800OutputTable: X800OutputTable =
  [[[new: x800OutStart, send: TRUE], -- x800OutStart, normal
    [new: x800OutGotDLE, send: TRUE], -- x800OutStart, dle
    [new: x800OutHeader, send: TRUE, zero: TRUE], -- x800OutStart, soh
    [new: x800OutText, send: TRUE, zero: TRUE], -- x800OutStart, stx
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutStart, etb (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutStart, itb (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutStart, etx (E)
    [new: x800OutStart, send: TRUE], -- x800OutStart, enq
    [new: x800OutStart, send: TRUE], -- x800OutStart, bel
    [new: x800OutStart, send: TRUE], -- x800OutStart, eot
    [new: x800OutStart, send: TRUE], -- x800OutStart, nak
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutStart, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutStart, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutStart, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutStart, ?
    [new: x800OutStart, send: TRUE, zero: TRUE]], -- x800OutStart, ?
   [[new: x800OutHeader, send: TRUE, crc: TRUE], -- x800OutHeader, normal
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, dle (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, soh (E)
    [new: x800OutText, send: TRUE, crc: TRUE], -- x800OutHeader, stx
    [new: x800OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
    -- x800OutHeader, etb
    [new: x800OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
    -- x800OutHeader, itb
```

```
    [new: x800OutBCC1, back: TRUE, send: TRUE, crc: TRUE],
    -- x800OutHeader, etx
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, enq (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, bel
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, eot (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, nak (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutHeader, ?
    [new: x800OutStart, send: TRUE, zero: TRUE]], -- x800OutHeader, ?
  [[new: x800OutText, send: TRUE, crc: TRUE], -- x800OutText, normal
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, dle (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, soh (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, stx (E)
    [new: x800OutBCC1, back: TRUE, send: TRUE, crc: TRUE], -- x800OutText, etb
    [new: x800OutBCC1, back: TRUE, send: TRUE, crc: TRUE], -- x800OutText, itb
    [new: x800OutBCC1, back: TRUE, send: TRUE, crc: TRUE], -- x800OutText, etx
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, enq (E)
    [new: x800OutText, send: TRUE, zero: TRUE], -- x800OutText, bel
    [new: x800OutText, send: TRUE, zero: TRUE], -- x800OutText, eot (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, nak (E)
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutText, ?
    [new: x800OutStart, send: TRUE, zero: TRUE]], -- x800OutText, ?
  [[new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, normal
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, dle
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, soh
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, stx
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, etb
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, itb
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, etx
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, enq
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, bel
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, eot
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, nak
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, ?
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, ?
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE], -- x800OutBCC1, ?
    [new: x800OutBCC2, back: TRUE, sendCRC: TRUE]], -- x800OutBCC1, ?
  [[new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, normal
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, dle
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, soh
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, stx
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, etb
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, itb
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, etx
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, enq
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, bel
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, eot
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, nak
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, ?
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, ?
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, ?
    [new: x800OutStart, sendCRC: TRUE], -- x800OutBCC2, ?
```



```
    [new: x800OutStart, sendCRC: TRUE]], -- x800OutBCC2, ?
  [[new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, normal
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, dle
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, soh
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, stx
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, etb
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, itb
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, etx
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, enq
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, bel
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, eot
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, nak
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, ?
    [new: x800OutStart, send: TRUE, zero: TRUE], -- x800OutGotDLE, ?
    [new: x800OutStart, send: TRUE, zero: TRUE]]]; -- x800OutGotDLE, ?
X800Init: PUBLIC PROCEDURE [pointer: LONG POINTER]
  RETURNS [inputStart, outputStart: MIOCInternalD0.State] =
  BEGIN
  charPtr: LONG ORDERED POINTER TO MIOCInternalD0.CharTable;
  inputPtr: LONG ORDERED POINTER TO X800InputTable;
  outputPtr: LONG ORDERED POINTER TO X800OutputTable;
  charPtr ← LOOPHOLE[pointer];
  charPtr↑ ← x800CharTable;
  inputPtr ← LOOPHOLE[charPtr + SIZE[MIOCInternalD0.CharTable]];
  inputPtr↑ ← x800InputTable;
  outputPtr ← LOOPHOLE[inputPtr + SIZE[X800InputTable]];
  outputPtr↑ ← x800OutputTable;
  RETURN[x800InStart, x800OutStart];
  END;

END. -- X800TablesD0
```

LOG

Time: August 30, 1979 2:38 PM By: Bill Danielson Action: Created file
Time: September 19, 1979 3:11 PM By: Bill Danielson Action: Combined common tables and adde
***d fill character tables.*
Time: October 4, 1979 1:11 PM By: Bill Danielson Action: Modified for D0 implementation.
Time: November 29, 1979 2:15 PM By: Bill Danielson Action: Broke off to just include X800 tables.
Time: February 13, 1980 2:57 PM By: Bill Danielson Action: Misc bug fixes.
Time: April 9, 1980 5:59 PM By: Bill Danielson Action: Reduce global frame size.

```
-- UtilitySpaceImpl.mesa (last edited by: Luniewski on: September 18, 1980 10:28 AM)
-- This version of SpaceImpl is for UtilityPilot. It simulates the operation of the Virtual Memory Man
**ager only for simple file spaces and for data spaces resident in memory. Minimal checking of ar
**guments is done. At present, some operations are unimplemented. If a client of UtilityPilot has
**some strong reason for needing them, they may be implemented in the future.
-- This module is an edited version of SpaceImpl.mesa. Changes to that module should track chan
**ges to this one.

DIRECTORY
  CachedRegion USING [Apply, Desc, kill, Operation, Outcome, writeProtect],
  CachedSpace USING [
    DataOrFile, Desc, Get, Handle, handleNull, handleVM, Level, Update],
  Environment USING [Long, maxPagesInMDS, PageCount],
  File USING [GetAttributes, GetSize, PageCount, Unknown, write],
  Frame USING [GetReturnLink, MyLocalFrame],
  Inline USING [BITAND, LowHalf],
  PrincOps USING [Frame, StateVector, TrapLink],
  Process USING [Detach],
  Runtime USING [CallDebugger],
  SDDefs USING [SD, sWriteProtect],
  SimpleSpace USING [Create, defaultCount, ForceOut, Map, Unmap],
  Space USING [
    defaultBase, defaultWindow, ErrorType, Handle, PageCount, PageNumber,
    PageOffset, WindowOrigin, wordsPerPage],
  SpecialSpace USING [],
  StoragePrograms USING [HandleFromPage, LongPointerFromPage],
  SwapperException USING [Await],
  SystemInternal USING [Unimplemented],
  Transaction USING [Handle],
  Trap USING [ReadOTP],
  VM USING [Interval, PageCount],
  VMMPrograms USING [],
  Volume USING [ID, Unknown];

UtilitySpaceImpl: MONITOR
  IMPORTS
    CachedRegion, CachedSpace, File, Frame, Inline, Process, Runtime, SimpleSpace,
    StoragePrograms, SwapperException, SystemInternal, Transaction, Trap, Volume
  EXPORTS SpecialSpace, Space, VMMPrograms
  SHARES File -- to extract permissions -- =
  BEGIN OPEN Space;
  Interval: TYPE = VM.Interval;
  Level: TYPE = CachedSpace.Level;
  SpaceD: TYPE = CachedSpace.Desc;
  RegionD: TYPE = CachedRegion.Desc;
  ----------------------------------------
  -- Space data:
  ----------------------------------------
  -- Public Constants
  Error: PUBLIC ERROR [type: ErrorType] = CODE;
  InsufficientSpace: PUBLIC ERROR [available: PageCount] = CODE;
  Handle: PUBLIC TYPE = CachedSpace.Handle;
  mds: PUBLIC Handle;
  nullHandle: PUBLIC Handle ← CachedSpace.handleNull;
  virtualMemory: PUBLIC Handle ← CachedSpace.handleVM;
  -- Private data:
  AddressFault: ERROR [page: PageNumber] = CODE; -- not to be caught by client
  InsufficientPermissions: ERROR [page: PageNumber] = CODE;
```



```
  -- not to be caught by client
  InternalError: ERROR [type: InternalErrorType] = CODE;
  -- not to be caught by client;
  InternalErrorType: TYPE = {bug0, bug1, bug2, bug3, bug4, bug5, bug6};
  intervalMDS: Interval;
  maxPagesInCodeBlock: PageCount = Environment.maxPagesInMDS;
  codeBdyMask: WORD = LOOPHOLE[-(maxPagesInCodeBlock - 1) - 1, WORD];
  -- = BITNOT[maxPagesInCodeBlock-1]
  ----------------------------------------
  -- Initialization:
  ----------------------------------------
  --VMMPrograms.--
  InitializeSpace: PUBLIC PROCEDURE [
    countVM: VM.PageCount, handleMDS: CachedSpace.Handle] =
    BEGIN
    -- countVM not used in UtilitySpaceImpl (or we would have a name collision!)
    mds ← handleMDS;
    intervalMDS ← [handleMDS.page, Environment.maxPagesInMDS];
    -- done differently than in SpaceImpl
    SDDefs.SD[SDDefs.sWriteProtect] ← WriteProtectTrap;
    Process.Detach[FORK VMMHelperProcess[]];
    END;
  ----------------------------------------
  -- SpecialSpace implementation:
  ----------------------------------------
  --Currently this is only used by UserTerminal; it is a candidate for flushing

  CreateForCode: PUBLIC ENTRY PROCEDURE [
    size: PageCount, parent: Handle, base: PageOffset] RETURNS [Handle] =
    -- Try it. If fails, possibly pad, then try again.
    BEGIN
    m: PageCount = Environment.maxPagesInMDS;
    h: Handle ← CreateInternal[size, parent, base];
    p: PageNumber ← VMPageNumber[h];
    IF size NOT IN (0..m] THEN RETURN WITH ERROR Error[invalidParameters];
    IF p/m = (p + size - 1)/m THEN RETURN[h];
    p ← (p + size) MOD m; -- next page to allocate in seg
    IF (p + size) > m THEN [] ← CreateInternal[m - p, parent, base];
    RETURN[CreateInternal[size, parent, base]];
    END;

  MakeResident, MakeSwappable: PUBLIC --ENTRY--PROCEDURE [space: Handle] = {
    -- all code and data is always resident under UtiliyPilot --};

  MakeCodeResident, MakeCodeSwappable: PUBLIC --ENTRY--PROCEDURE [
    frame: PROGRAM] = {
    -- all code and data is always resident under UtiliyPilot --};
  ----------------------------------------
  -- Space implementation:
  ----------------------------------------

  Activate, Deactivate, DeleteSwapUnits: PUBLIC --ENTRY--PROCEDURE [
    space: Handle] = {};

  CopyIn, CopyOut: PUBLIC --ENTRY--PROCEDURE [
    space: Handle, window: WindowOrigin, transaction: Transaction.Handle] = {
    SIGNAL SystemInternal.Unimplemented; };
```

```
Create: PUBLIC ENTRY PROCEDURE [
  size: PageCount, parent: Handle, base: PageOffset] RETURNS [Handle] = {
  RETURN[CreateInternal[size: size, parent: parent, base: base]]};

CreateInternal: INTERNAL PROCEDURE [
  size: PageCount, parent: Handle, base: PageOffset] RETURNS [Handle] =
  -- parent must be virtualMemory (no nested spaces allowed).
  -- base must be defaultBase.
  BEGIN
  IF size = 0 OR base ~= defaultBase OR
    ~(parent = CachedSpace.handleVM OR parent = mds) THEN
    RETURN WITH ERROR Error[invalidParameters];
  RETURN[
    SimpleSpace.Create[
      size,
      IF parent = CachedSpace.handleVM THEN hyperspace ELSE --parent = mds--mds]];
  END;
-- This is not a perfect simulation since GetAttributes will return wrong data.

CreateUniformSwapUnits: PUBLIC --ENTRY--PROCEDURE [
  size: PageCount, parent: Handle] = {};

Delete: PUBLIC --ENTRY--PROCEDURE [space: Handle] =
  BEGIN
  spaceD: SpaceD;
  CachedSpace.Get[@spaceD, space];
  IF spaceD.state = mapped THEN
    -- recovers real memory. (Spaces created by StartPilot may not look like SimpleSpaces.)
    BEGIN
    spaceD.dPinned ← TRUE;
    [] ← CachedSpace.Update[@spaceD];
    SimpleSpace.Unmap[space];
    END;
  -- (can't actually delete simpleSpaces.)

  END;

ForceOut: PUBLIC ENTRY PROC [space: Handle] = {SimpleSpace.ForceOut[space]; };

GetAttributes: PUBLIC --ENTRY--PROCEDURE [space: Handle]
  RETURNS [
    parent, lowestChild, nextSibling: Handle, base: PageOffset, size: PageCount,
    mapped: BOOLEAN] = {SIGNAL SystemInternal.Unimplemented; };

GetHandle: PUBLIC --ENTRY--PROCEDURE [page: PageNumber] RETURNS [Handle] = {
  RETURN[StoragePrograms.HandleFromPage[page]]; };

GetWindow: PUBLIC ENTRY PROCEDURE [space: Handle] RETURNS [WindowOrigin] =
  BEGIN
  spaceD: SpaceD;
  CachedSpace.Get[@spaceD, space];
  IF spaceD.state ~= mapped THEN RETURN WITH ERROR Error[noWindow];
  RETURN[IF spaceD.dataOrFile = file THEN spaceD.window ELSE defaultWindow]
  END;

Kill: PUBLIC ENTRY PROCEDURE [space: Handle] =
  BEGIN [] ← CachedRegion.Apply[space.page, CachedRegion.kill]; END;
```



```
LongPointer: PUBLIC PROCEDURE [space: Handle] RETURNS [LONG POINTER] =
  BEGIN RETURN[StoragePrograms.LongPointerFromPage[space.page]]; END;

LongPointerFromPage: PUBLIC PROCEDURE [page: PageNumber]
  RETURNS [LONG POINTER] = {
  RETURN[StoragePrograms.LongPointerFromPage[page]]; };

MakeReadOnly: PUBLIC ENTRY PROCEDURE [
  space: Handle, transaction: Transaction.Handle] =
  BEGIN
  spaceD: SpaceD;
  CachedSpace.Get[@spaceD, space];
  spaceD.writeProtected ← TRUE;
  [] ← CachedSpace.Update[@spaceD];
  [] ← CachedRegion.Apply[space.page, CachedRegion.writeProtect];
  END;

Map: PUBLIC ENTRY PROCEDURE [
  space: Handle, window: WindowOrigin, transaction: Transaction.Handle] =
  BEGIN
  countMapped: Space.PageCount;
  spaceD: SpaceD;
  volumeUnknown: Volume.ID;
  CachedSpace.Get[@spaceD, space];
  BEGIN
  ENABLE
    BEGIN
    File.Unknown => GO TO UnknownFile;
    Volume.Unknown --[volume]-- =>
      BEGIN volumeUnknown ← volume; GO TO UnknownVolume END
    END;
  IF window.file = defaultWindow.file THEN
    BEGIN --data space--countMapped ← spaceD.interval.count; END
  ELSE
    BEGIN --file space--
    countFile: File.PageCount =
      MAX[File.GetSize[window.file], window.base] - window.base;
    countMapped ←
      IF spaceD.interval.count <= countFile THEN spaceD.interval.count
      ELSE Inline.LowHalf[countFile]; -- no danger of truncation

    END;
  EXITS
    UnknownFile => RETURN WITH ERROR File.Unknown[window.file];
    UnknownVolume => RETURN WITH ERROR Volume.Unknown[volumeUnknown];
  END;
  SimpleSpace.Map[
    handle: space, window: window, andPin: FALSE, countMapped: countMapped];
  IF window.file ~= defaultWindow.file AND
    (File.GetAttributes[window.file].immutable OR Inline.BITAND[
      window.file.permissions, File.write] = 0) THEN
    BEGIN
    CachedSpace.Get[@spaceD, space];
    spaceD.writeProtected ← TRUE;
    [] ← CachedSpace.Update[@spaceD];
    END;
  END;

PageFromLongPointer: PUBLIC --ENTRY--PROCEDURE [lp: LONG POINTER]
```

```
  RETURNS [page: PageNumber] =
  BEGIN OPEN LOOPHOLE[lp, num Environment.Long];
  IF lp = NIL THEN ERROR Error[invalidParameters]
  ELSE RETURN[highbits*256 + lowbits/256];
  END;

Pointer: PUBLIC ENTRY PROCEDURE [space: Handle] RETURNS [POINTER] =
  BEGIN
  spaceD: SpaceD;
  CachedSpace.Get[@spaceD, space];
  IF spaceD.state = missing THEN RETURN WITH ERROR Error[invalidHandle];
  IF space.level <= mds.level OR space.page ~IN
    [intervalMDS.page..intervalMDS.page + intervalMDS.count) THEN
    RETURN WITH ERROR Error[invalidParameters];
  RETURN[LOOPHOLE[(space.page - intervalMDS.page)*wordsPerPage]]
  END;

Remap: PUBLIC --ENTRY--PROCEDURE [
  space: Handle, window: WindowOrigin, transaction: Transaction.Handle] = {
  SIGNAL SystemInternal.Unimplemented; };

Unmap: PUBLIC --ENTRY--PROCEDURE [space: Handle] = {SimpleSpace.Unmap[space]; };

VMPageNumber: PUBLIC --ENTRY--PROCEDURE [space: Handle] RETURNS [PageNumber] = {
  RETURN[space.page]};
-- Other

VMMHelperProcess: PROCEDURE = -- root of VMM helper process, a monitor external
  BEGIN
  page: PageNumber;
  operation: CachedRegion.Operation;
  outcome: CachedRegion.Outcome;
  HandleException: PROCEDURE = INLINE
    BEGIN
    WITH outcome SELECT FROM
      ok => RETURN;
      error --[state]-- =>
        IF operation.action = activate THEN Runtime.CallDebugger["AddressFault"]
        ELSE ERROR;
      ENDCASE --regionDDirty, regionDMissing, spaceDMissing-- => ERROR;
    END;
  DO
    --FOREVER--
    [page, operation, outcome] ← SwapperException.Await[];
    HandleException[];
    ENDLOOP;
  END;

WriteProtectTrap: PROCEDURE =
  -- handler for processor-generated trap, independent of monitor
  BEGIN
  page: PageNumber;
  state: RECORD [a: UNSPECIFIED, v: PrincOps.StateVector];
  state.v ← STATE;
  state.v.dest ← Frame.GetReturnLink[];
  state.v.source ← PrincOps.TrapLink;
  -- when microcode settles down, remove redundant assignment to myLocalFrame.unused
  page ← LOOPHOLE[Frame.MyLocalFrame[], POINTER TO local PrincOps.Frame].unused
    ← LOOPHOLE[Trap.ReadOTP[], PageNumber]; -- trap parameter
```



```
    DO Runtime.CallDebugger["WriteProtectFault"]; ENDLOOP;
    END;

  END.
```

LOG

Time: September 12, 1979 10:36 AM By: Knutsen Action: Created file from SpaceIm
***pl of August 30, 1979. Added InitSpace.*
Time: October 22, 1979 10:41 AM By: Knutsen Action: Changed Forgot to Syste
***mInternal.*
Time: January 28, 1980 11:48 AM By: Forrest Action: Copied LongPtr< = >Page
***code in from SpaceImpl; added export of StoragePrograms; Merged in InitVMMgr from UtilityV*
***MMControl (now dead)*
Time: March 10, 1980 5:05 PM By: Forrest Action: Made CreateUniformSwapUnits be a
***no-op*
Time: April 17, 1980 1:32 PM By: Knutsen Action: Added InitializeSpace. Changes for
***non-zero MDS. Add transaction Handle. Implement Delete.*
Time: April 23, 1980 10:29 AM By: Knutsen Action: Made Delete be a no-op.
Time: April 28, 1980 11:59 AM By: Forrest/Dale Action: FrameOps = >Frame, ControlDefs = >
***PrincOps, TrapOps = >Trap. Implement Delete. Add updating of Cache in Delete.*
Time: June 27, 1980 10:31 AM By: McJones Action: Add transaction handle to MakeRead
***Only, Remap; remove interrupt disable in trap handler*
Time: September 18, 1980 10:28 AM By: Luniewski Action: Add CopyIn and CopyOut
***as "SystemInternal.Unimplemented" procedures.*

```
-- UtilityStore>UtilityVMMControl.mesa (last edited by Forrest on September 3, 1980 10:39 AM)
-- This version of VMMControl is for UtilityPilot. Its compelling reason for existence is that UtilitySp
**aceImpl can not export StoragePrograms due to a name conflict with Space.LongPointerFromP
**age, etc.
-- This module is an edited version of VMMControl.mesa. Changes to this module should track cha
**nges to it.

DIRECTORY
  CachedSpace USING [Handle, Level],
  Environment USING [PageCount, PageNumber],
  PerformancePrograms USING [],
  RuntimePrograms USING [],
  TransactionState USING [FileOps, LogEntryPtr],
  StoragePrograms USING [pageMDS],
  Transaction USING [Handle],
  VMMPrograms USING [InitializeSpace];

UtilityVMMControl: PROGRAM
  IMPORTS StoragePrograms, VMMPrograms
  EXPORTS
    PerformancePrograms, RuntimePrograms, StoragePrograms, Transaction,
    TransactionState = PUBLIC

  BEGIN
  --PerformancePrograms.--
  InitializePilotCounter: PROC = {};
  --PerformancePrograms.--

  InitializePilotPerfMonitor: PROC = {};
  --RuntimePrograms.--

  InitializePilotLoadState: PROC [Environment.PageNumber, Environment.PageCount] =
    {};
  --StoragePrograms.--

  InitializeTransactionData: PROC = {};
  --StoragePrograms.--

  InitializeVMMgr: PROC [
    countVM: Environment.PageCount, pMapLogDesc: LONG POINTER] =
    BEGIN
    mds: CachedSpace.Handle =
      [level: FIRST[CachedSpace.Level] + 1, page: StoragePrograms.pageMDS];
    VMMPrograms.InitializeSpace[countVM: countVM, handleMDS: mds]
    END;
  --StoragePrograms.--

  IsDiagnosticPilot: PROC RETURNS [BOOLEAN] = {RETURN[FALSE]};
  --StoragePrograms.--

  IsUtilityPilot: PROC RETURNS [BOOLEAN] = {RETURN[TRUE]};
  --StoragePrograms.--

  RecoverTransactions: PROC = {};
  --Transaction.--

  nullHandle: PUBLIC Transaction.Handle ← LOOPHOLE[0]; -- for FileImpl.
  --TransactionState.--
```



```
  Log: PROC [
    Transaction.Handle, TransactionState.LogEntryPtr, TransactionState.FileOps] =
    {}; -- for FileImpl.

  END.
```

LOG

April 17, 1980 1:23 PM	Knutsen	Created file from VMMPrograms.mesa of April 10, 1980 **4:35 PM.
May 1, 1980 10:15 AM	Forrest	Added Stubs fro RuntimePerf.
July 7, 1980 12:37 PM	Luniewski	Made InitializePilotLoadState take arguments.
July 15, 1980 3:43 PM	Gobbel	Added stubs for Transaction, TransactionInternal.
August 11, 1980 10:55 AM	Knutsen	Converted to latest Transaction stuff.
August 28, 1980 5:15 PM	Knutsen	SpecialTransaction renamed to TransactionState. Thr **ew out all now-obsolete Transaction stuff.
September 3, 1980 10:29 AM	Forrest	Recovered file from Dale's directory. Deleted PPD*.

-- Note: Unused facilities in MStoreImpl, ResidentMemoryImpl, SimpleSpaceImpl, and CachedRegionImplA, as well as those in Processes and Signals, can be moved into an unused swap unit when the Packager arrives.

```
PACK
    BootSwapCross,
    CachedRegionImplA,
    DiagnosticPilotImpl,
    DiagnosticPilotStubsA,
    Instructions,
    MStoreImpl,
    PilotControl,
    PilotNub,
    Processes,
    ResidentMemoryImpl,
    SetGMTUsingEthernet,
    Signals,
    SimpleSpaceImpl,
    StartChainPlug,
    SystemImpl,
    Traps;

DiagnosticPilot: CONFIGURATION
    IMPORTS DiagnosticPilotClient,
        -- device faces -- EthernetOneFace, HeadStartChain, KeyboardFace, ProcessorFace
    EXPORTS DeviceCleanup, PilotSwitches, ProcessInternal, Process, ResidentMemory, Runtime, RuntimeInternal, SpecialSpace,
        System =

BEGIN

    -- "DiagnosticControl"
PilotControl; -- (imports DriverStartChain, HeadStartChain, StoreDriverStartChain.)
DiagnosticPilotImpl;
DiagnosticPilotStubsA;
SystemImpl;

    -- "DiagnosticStore"
CachedRegionImplA; -- (contains AllocateMStoreRuthlessly for ResidentMemory)
MStoreImpl;
SimpleSpaceImpl; -- (VM allocator)
ResidentMemoryImpl;

    -- "DiagnosticMesaRuntime"
BootSwapCross;
Instructions;
PilotNub;
Processes; -- need mini Processes with Aborted, Detach, GetPriority, MsecToTicks, SetPriority, SetTimeout (no Fork/Join/PSB pool).
Signals; -- need a mini Signals which always goes straight to debugger.
Traps;

    -- Temporary,
SetGMTUsingEthernet;
[DriverStartChain, DontCare, StoreDriverStartChain] ← StartChainPlug[]; -- stop driver and store driver start chains from PilotControl

END.
```

LOG

(For earlier log entries see Pilot 4.0 archive version.)

Time: April 16, 1980 6:42 PM	*By: McJones*	*Action: Add SetGMTUsingEthernet*
Time: April 16, 1980 11:27 PM	*By: Forrest*	*Action: Add EthernetHeadD0 and UserTerminalHeadD0 to PACK*
Time: April 19, 1980 4:46 PM	*By: Knutsen*	*Action: Add DiagnosticPilotImpl, DiagnosticPilotStubA, MStoreImpl, ResidentMemoryImpl, SimpleSpaceImpl, CachedRegionImplA; DiagnosticPilotControl replaced by PilotControl*
Time: May 19, 1980 1:16 PM	*By: McJones*	*Action: Define OISProcessorFace with THEN to avoid conflict with HeadStartChain.Start*
Time: June 18, 1980 8:41 AM	*By: McJones*	*Action: Export ResidentMemory, RuntimeInternal, Process*
Time: July 22, 1980 4:24 PM	*By: Fay*	*Action: Delete OIS- prefix from all OISProcessor...; correct all module assignment statements*
Time: August 6, 1980 10:36 AM	*By: Sandman*	*Action: ?*
Time: August 20, 1980 10:06 AM	*By: McJones*	*Action: Remove heads*
Time: August 29, 1980 10:57 PM	*By: Forrest*	*Action: Make current*

```
-- DiagnosticPilot>DiagnosticPilotStubsA.mesa (last edited by Forrest on July 20, 1980 9:29 PM)
-- The compelling reason for the existence of this module is that DiagnosticPilotImpl can not expor
**t two different procedures to two different interfaces if the procedures have the same name.

DIRECTORY
  CachedSpace,
  PrincOps USING [ControlLink, GlobalFrameHandle],
  PrincOpsRuntime USING [GetFrame, GFT],
  Runtime USING [],
  SDDefs USING [SD, sGFTLength],
  SystemInternal USING [Unimplemented],
  Transaction USING [];

DiagnosticPilotStubsA: PROGRAM
  IMPORTS PrincOpsRuntime, SystemInternal
  EXPORTS CachedSpace, Runtime, Transaction
  SHARES Transaction =
  BEGIN
  --CachedSpace.--
  Get: PUBLIC PROC [pDescResult: CachedSpace.PDesc, space: CachedSpace.Handle] =
    BEGIN SIGNAL SystemInternal.Unimplemented END; -- for SimpleSpaceImpl.
  -- Name conflict with CachedRegion.Insert:
  --CachedSpace.--

  Insert: PUBLIC PROC [pDescVictim: CachedSpace.PDesc, pDesc: CachedSpace.PDesc] =
    BEGIN END; -- for SimpleSpaceImpl.DescribeSpace.
  --CachedSpace.--

  Update: PUBLIC PROC [pDesc: CachedSpace.PDesc] RETURNS [found: BOOLEAN] =
    BEGIN SIGNAL SystemInternal.Unimplemented END; -- for SimpleSpaceImpl.
  -- Name conflict with Space.nullHandle:
  --Transaction.--

  Handle: PUBLIC TYPE = CARDINAL; -- any one-word type would do
  --Transaction.--
  nullHandle: PUBLIC Handle ← 0;
  -- Start trap needs ValidateGlobalFrame:
  InvalidGlobalFrame: PUBLIC ERROR [frame: PrincOps.GlobalFrameHandle] = CODE;
  --Runtime.--
  ValidateGlobalFrame: PUBLIC PROC [g: PrincOps.GlobalFrameHandle] =
    BEGIN OPEN LOOPHOLE[g, rep PrincOps.ControlLink];
    IF proc OR indirect OR ~InGFT[g] THEN ERROR InvalidGlobalFrame[g];
    END;

  InGFT: PROC [g: PrincOps.GlobalFrameHandle] RETURNS [BOOLEAN] =
    BEGIN OPEN PrincOpsRuntime;
    FOR i: CARDINAL IN [0..SDDefs.SD[SDDefs.sGFTLength]) DO
      IF GetFrame[GFT[i]] = g AND g.gfi = i THEN RETURN[TRUE]; ENDLOOP;
    RETURN[FALSE];
    END;

  END.
LOG
Time: April 20, 1980 4:13 PM    By: Knutsen     Action: Created file.
Time: May 27, 1980 4:56 PM      By: Forrest     Action: Add Validate GlobalFrame, InitializeF
**rames (now needed by Traps).
Time: July 20, 1980 1:16 PM     By: Forrest     Action: CHange to use PrincOpsRuntime
```

```
-- DiagnosticPilotImpl.mesa (last edited by: Forrest on: August 29, 1980 5:24 PM)

DIRECTORY
  Boot USING [Location, LVBootFiles],
  CachedRegion USING [Operation, Outcome],
  CachedSpace USING [handleNull],
  CommunicationPrograms USING [],
  Device USING [Type],
  DiagnosticPilotClient USING [Run],
  Environment USING [PageCount, PageNumber],
  File USING [Capability, PageCount, PageNumber],
  KernelFile USING [],
  MiscPrograms USING [],
  MStore USING [AllocateIfFree, Deallocate],
  PerformancePrograms USING [],
  PilotClient USING [],
  PrincOps USING [NullLink],
  ProcessorFace USING [BootButton],
  Runtime USING [CallDebugger],
  RuntimePrograms USING [],
  SDDefs USING [SD, sPageFault, sWriteProtect, sCopy, sUnNew],
  SimpleSpace USING [AllocateVM],
  Space USING [Handle, PageCount, PageOffset, WindowOrigin],
  SpecialSpace USING [],
  StoragePrograms USING [],
  SystemInternal USING [Unimplemented],
  TemporaryBooting USING [Switches],
  Transaction USING [Handle],
  VM USING [Interval, PageNumber],
  Volume USING [ID, nullID, TypeSet];

DiagnosticPilotImpl: PROGRAM
  IMPORTS
    DiagnosticPilotClient, MStore, ProcessorFace, Runtime, SimpleSpace,
    SystemInternal
  EXPORTS
    CachedRegion, CommunicationPrograms, KernelFile, MiscPrograms,
    PerformancePrograms, PilotClient, RuntimePrograms, Space, SpecialSpace,
    StoragePrograms, TemporaryBooting, Volume
  SHARES Transaction =
  BEGIN
  pagesResidentMemory: Environment.PageCount = 20; -- for frame allocation, etc.
  moreThanMaxRealMemSize: Environment.PageCount = 10000; --
  -- StoragePrograms
  IsDiagnosticPilot: PUBLIC PROC RETURNS [BOOLEAN] = {RETURN[TRUE]};

  IsUtilityPilot: PUBLIC PROC RETURNS [BOOLEAN] = {RETURN[FALSE]}; --
  -- PilotClient

  Run: PUBLIC PROC =
    BEGIN
    pageWorkArea: Environment.PageNumber;
    countWorkArea, countUserWorkArea: Environment.PageCount; -- real memory.
    pageWorkArea ← SimpleSpace.AllocateVM[moreThanMaxRealMemSize, hyperspace];
    countWorkArea ← MStore.AllocateIfFree[
      VM.Interval[pageWorkArea, moreThanMaxRealMemSize]];
    -- find size of available real memory.
    MStore.Deallocate[
      interval: VM.Interval[pageWorkArea, countWorkArea], promised: FALSE];
```



```
    countUserWorkArea ← MStore.AllocateIfFree[
      VM.Interval[pageWorkArea, countWorkArea - pagesResidentMemory]];
    BEGIN OPEN SDDefs;
    IF SD[sPageFault] = PrincOps.NullLink THEN SD[sPageFault] ← PageFaultPlug;
    IF SD[sWriteProtect] = PrincOps.NullLink THEN
      SD[sWriteProtect] ← WriteProtectPlug;
    IF SD[sCopy] = PrincOps.NullLink THEN SD[sCopy] ← NEWPlug;
    IF SD[sUnNew] = PrincOps.NullLink THEN SD[sUnNew] ← UnNewPlug;
    END;
    DiagnosticPilotClient.Run[pageWorkArea, countUserWorkArea]; -- never returns.

    END; ------------------------------------------
  -- Substitute implementations of Pilot facilities:
  ------------------------------------------
  --
  -- CachedRegion

  Apply: PUBLIC PROC [
    pageMember: VM.PageNumber, operation: CachedRegion.Operation]
    RETURNS [outcome: CachedRegion.Outcome, pageNext: VM.PageNumber] = {
    SIGNAL SystemInternal.Unimplemented}; -- for SimpleSpaceImpl.
  --
  -- Space

  nullHandle: PUBLIC Space.Handle ← LOOPHOLE[CachedSpace.handleNull];
  -- for PilotControl.
  Kill: PUBLIC PROCEDURE [Space.Handle] = {};
  -- this could be smarter & take real memory & give it to the client.
  --
  -- TemporaryBooting

  BootButton: PUBLIC PROC [switches: TemporaryBooting.Switches] = {
    ProcessorFace.BootButton[]}; -- for PilotNub, Traps.
  --
  -- Volume

  Close: PUBLIC PROC [Volume.ID] = {SIGNAL SystemInternal.Unimplemented};
  -- for SystemImpl.

  GetNext: PUBLIC PROC [Volume.ID, Volume.TypeSet] RETURNS [Volume.ID] = {
    RETURN[Volume.nullID]}; -- for SystemImpl.
  --
  -- Miscellaneous SD slots

  PageFaultPlug: PROC = {Runtime.CallDebugger["Page fault trap"L]};

  WriteProtectPlug: PROC = {Runtime.CallDebugger["Write protect trap"L]};

  NEWPlug: PROC = {Runtime.CallDebugger["NEW not implemented"L]};

  UnNewPlug: PROC = {Runtime.CallDebugger["UnNew not implemented"L]};
  ------------------------------------------
  -- Dummy implementations of unneeded Pilot facilities (for PilotControl):
  ------------------------------------------
  --
  -- CommunicationPrograms

  InitializeCommunication: PUBLIC PROC = {}; --
```

```
  -- KernelFile

  Pin: PUBLIC PROC [
    file: File.Capability, page: File.PageNumber, count: File.PageCount] = {}; --
  -- MiscPrograms

  InitializeHeap, InitializeResidentHeap, InitializeStream, InitializeUtilities,
    InitializeZone: PUBLIC PROC = {}; --
  -- PerformancePrograms

  InitializePilotCounter, InitializePilotPerfMonitor: PUBLIC PROC = {}; --
  -- RuntimePrograms

  InitializeFrames, InitializeSnapshot: PUBLIC PROC = {};

  InitializePilotLoadState: PUBLIC PROC [
    pageInitialLoadState: Environment.PageNumber,
    countInitialLoadState: Environment.PageCount] = {}; --
  -- Space

  Create: PUBLIC PROC [
    size: Space.PageCount, parent: Space.Handle, base: Space.PageOffset]
    RETURNS [h: Space.Handle] = {RETURN[h]};
  -- The following item has a name conflict with CachedSpace.Delete:

  Delete: PUBLIC PROC [space: Space.Handle] = {};

  Map: PUBLIC PROC [
    space: Space.Handle, window: Space.WindowOrigin,
    transaction: Transaction.Handle] = {}; --
  -- SpecialSpace

  MakeSwappable: PUBLIC PROC [space: Space.Handle] = {}; --
  -- StoragePrograms

  InitializeFileMgr: PUBLIC PROC [
    bootFile: LONG POINTER TO disk Boot.Location,
    pLVBootFiles: POINTER TO Boot.LVBootFiles]
    RETURNS [debuggerDeviceType: Device.Type, debuggerDeviceOrdinal: CARDINAL] = {
    RETURN[debuggerDeviceType, debuggerDeviceOrdinal]};

  InitializeFiler: PUBLIC PROC = {};

  InitializeSwapper: PUBLIC PROC [pMapLogDesc: LONG POINTER] = {};

  InitializeTransactionData: PUBLIC PROC = {};

  InitializeVMMgr: PUBLIC PROC [
    countVM: Environment.PageCount, pMapLogDesc: LONG POINTER] = {};

  RecoverTransactions: PUBLIC PROC = {};

  ReplacementProcess: PUBLIC PROC [threshold: Environment.PageCount] = {};

  END....
LOG
Time: April 20, 1980  4:12 PM    By: Knutsen    Action: Create file
Time: May 17, 1980  8:39 PM      By: Forrest    Action: ?
```



```
Time: June 14, 1980  5:50 PM    By: Knutsen    Action: Plug up unused SD slots
Time: June 25, 1980  4:48 PM    By: McJones    Action: OISProcessorFace = >ProcessorFac
**e; delete Space.nullTransactionhandle; add InitializeHeap; add parameters to InitializePilotLoad
**State
Time: July 20, 1980  10:21 AM   By: Forrest    Action: Add second parameter to GetNext
Time: July 22, 1980  5:30 PM    By: Fay    Action: Add dummy stubs for Space.Kill, StorageProgr
**ams.InitializeTransactionData, and StoragePrograms.RecoverTransactions.
Time: August 17, 1980  12:18 PM    By: Fay    Action: VolumeSet = > TypeSet.
```

```
-- BootChannelDisk.mesa (last edited by: McJones on: August 23, 1980 1:21 PM)

DIRECTORY
  Boot USING [Location, LP],
  BootChannel USING [Create, Operation, Handle],
  DeviceTypes USING [sa4000],
  Environment USING [PageCount, PageNumber],
  Inline USING [LowHalf],
  PilotDisk USING [Address, Label, SetLabelFilePage],
  PilotMP USING [cGermDeviceError, cGermLabelCheck, Code],
  SA4000Face USING [
    Command, DeviceHandle, DiskAddress, GetNextDevice, globalStateSize,
    Initialize, Initiate, nullDeviceHandle, Operation, operationSize, Poll,
    Recalibrate];

BootChannelDisk: PROGRAM
  IMPORTS Boot, RemainingChannels: BootChannel, Inline, PilotDisk, SA4000Face
  EXPORTS BootChannel
  SHARES PilotDisk =

  -- Implementation of BootChannel for Pilot volume on SA4000.
  -- A single, serially reusable, channel is supported.
  BEGIN OPEN Boot, Environment;

  daNull: PilotDisk.Address = [0, 0];
  label: PilotDisk.Label; -- HOW SHOULD THIS BE ALIGNED FOR FUTURE DISKS?
  pAllocateNext: LONG POINTER TO UNSPECIFIED; -- allocator for first 64K storage
  pOperationState: LONG POINTER TO SA4000Face.Operation;
  pLoc: POINTER TO Location;
  tries: [0..triesMax];
  triesMax: CARDINAL = 8;

  Create: PUBLIC PROCEDURE [
    pLocation: POINTER TO Location, operation: BootChannel.Operation,
    dFirst64KStorage: LONG DESCRIPTOR FOR ARRAY OF WORD]
    RETURNS [BootChannel.Handle] =
    BEGIN
    pAllocateNext ← BASE[dFirst64KStorage]; -- reset first 64K allocator
    SELECT (pLoc ← pLocation).deviceType FROM
      = DeviceTypes.sa4000 =>
        BEGIN
        da: SA4000Face.DiskAddress = LOOPHOLE[pLocation.diskFileID.da];
        pOp: LONG POINTER TO SA4000Face.Operation =
          (pOperationState ← Allocate[SA4000Face.operationSize]);
        device: SA4000Face.DeviceHandle;
        SA4000Face.Initialize[
          0, Inline.LowHalf[Allocate[SA4000Face.globalStateSize]]];
        device ← SA4000Face.nullDeviceHandle;
        THROUGH [0..pLocation.deviceOrdinal] DO
          device ← SA4000Face.GetNextDevice[device] ENDLOOP;
        -- Initialize label
        pOp.clientHeader ← da;
        pOp.labelPtr ← @label;
        -- dataPtr, incrementDataPtr irrelevant
        pOp.command ← vr; -- read label (and ignore data)
        -- pageCount set in Transfer
        pOp.device ← device;
        TransferSA4000[page: NULL, count: 1]; -- fetch type, attribute fields
```



```
        label.fileID ← pLoc.diskFileID.fID;
        PilotDisk.SetLabelFilePage[@label, pLoc.diskFileID.firstPage];
        label.bootChainLink ← daNull;
        --- so TransferSA4000 will use pOp.clientHeader
        -- Initialize remaining operation fields
        pOp.clientHeader ← da; -- since transfer above incremented it
        -- labelPtr set above
        -- dataPtr set in Transfer
        pOp.incrementDataPtr ← TRUE;
        pOp.command ←
          SELECT operation FROM
            read => vvr,
            write => vvw,
            ENDCASE --rawRead-- => vrr;
        -- pageCount set in Transfer
        -- device set above
        RETURN[TransferSA4000];
        END;
      ENDCASE => --not anything I implement. Pass it on.
        RETURN[RemainingChannels.Create[pLocation, operation, dFirst64KStorage]];
    END;

  TransferSA4000: PROCEDURE [page: PageNumber, count: PageCount] =
    BEGIN
    pOp: LONG POINTER TO SA4000Face.Operation = pOperationState;
    IF count = 0 THEN RETURN; -- transfers done; no cleanup necessary
    pOp.dataPtr ← LPFromPage[page];
    pOp.pageCount ← count;
    tries ← triesMax;
    DO
      -- until count pages transferred
      IF label.bootChainLink ~= daNull THEN
        BEGIN
        pOp.clientHeader ← LOOPHOLE[label.bootChainLink];
        label.bootChainLink ← daNull;
        END;
      count ← pOp.pageCount;
      SA4000Face.Initiate[pOp];
      DO
        -- until status~ = inProgress
        SELECT SA4000Face.Poll[pOp] FROM
          inProgress => NULL;
          goodCompletion => GO TO TransferComplete;
          labelCheck =>
            IF label.bootChainLink ~= daNull THEN GO TO OperationComplete
              -- end of run

            ELSE Error[PilotMP.cGermLabelCheck];
          ENDCASE -- other error -- =>
            BEGIN
            IF pOp.pageCount ~= count THEN tries ← triesMax - 1
            ELSE
              SELECT tries ← tries - 1 FROM
                0 => Error[PilotMP.cGermDeviceError];
                triesMax/2 => SA4000Face.Recalibrate[pOp.device];
                ENDCASE;
            GO TO OperationComplete
            END;
        REPEAT OperationComplete => NULL;
```

```
      ENDLOOP;
    REPEAT TransferComplete => NULL;
    ENDLOOP;
  END;

-- Allocator for 16-word aligned storage in first64K

Allocate: PROCEDURE [size: CARDINAL] RETURNS [lp: LONG POINTER TO UNSPECIFIED] =
  BEGIN pAllocateNext ← (lp ← pAllocateNext) + LONG[((size + 15)/16)*16] END;

Error: SIGNAL [PilotMP.Code] = CODE;

LPFromPage: PROCEDURE [page: PageNumber] RETURNS [LONG POINTER] = INLINE
  BEGIN RETURN[LP[highbits: page/256, lowbits: page*256 --MOD 2**16--]] END;

END.
```

(For earlier log entries see Pilot 4.0 archive version.)

June 23, 1980 2:31 PM	*McJones*	*OISDisk,FilePageLabel = >PilotDisk*
August 13, 1980 6:02 PM	*McJones*	*Read first label to determine attributes*
August 23, 1980 1:21 PM	*McJones*	*Don't report labelCheck at end of run*

```
-- BootChannelEther.mesa (last edited by: McJones on: June 23, 1980 2:37 PM)

DIRECTORY
  Boot USING [Location, LP],
  BootChannel USING [Create, Operation, Handle],
  DeviceTypes USING [ethernet],
  Environment USING [bytesPerPage, PageCount, PageNumber, wordsPerPage],
  MiniEthernetDefs USING [ActivateDriver, SendPacket, RecvPacket, KillDriver],
  ProcessorFace USING [GetClockPulses],
  PilotMP USING [
    cGermDeviceError, cGermFunnyPacket, cGermNoServer, cGermTimeout, Code],
  PupTypes USING [PupAddress, PupSocketID, PupType, Pair, miscSrvSoc],
  ResidentMemory USING [AllocateMDS];

BootChannelEther: PROGRAM
  IMPORTS
    Boot, RemainingChannels: BootChannel, MiniEthernetDefs, ProcessorFace,
    ResidentMemory
  EXPORTS BootChannel =
  -- Implementation of BootChannel for Ethernet 1.
  -- A single, serially reusable, channel is supported.
  BEGIN OPEN Boot, Environment;
  CantActivateDriver: PilotMP.Code = PilotMP.cGermDeviceError;
  NoBootServerResponded: PilotMP.Code = PilotMP.cGermNoServer;
  NextPacketDidntArrive: PilotMP.Code = PilotMP.cGermTimeout;
  FunnySizePacket: PilotMP.Code = PilotMP.cGermFunnyPacket;
  FunnySequenceNumber: PilotMP.Code = PilotMP.cGermFunnyPacket;
  pAllocateNext: LONG POINTER TO UNSPECIFIED; -- allocator for items in first 64K.
  bfn: CARDINAL; -- boot file number.
  iocb: LONG POINTER;
  bufferLength: CARDINAL = 300;
  fudge: CARDINAL = 25; -- must be bigger than encapsulation and pup overhead
  totalBufferLength: CARDINAL = bufferLength + fudge;
  buffer: LONG POINTER TO ARRAY (0..totalBufferLength] OF WORD;
  countTotalBuffer: PageCount =
    (totalBufferLength + wordsPerPage - 1)/wordsPerPage;

  mySocket: PupTypes.PupSocketID = LOOPHOLE[ProcessorFace.GetClockPulses[]];
  him: PupTypes.PupAddress;
  anyBootServer: PupTypes.PupAddress = [[0], [0], PupTypes.miscSrvSoc];

  recvStarted: BOOLEAN;
  receiveSeqNumber: CARDINAL;
  Create: PUBLIC PROCEDURE [
    pLocation: POINTER TO Location, operation: BootChannel.Operation,
    dFirst64KStorage: LONG DESCRIPTOR FOR ARRAY OF WORD]
    RETURNS [handle: BootChannel.Handle] =
    BEGIN
    pAllocateNext ← BASE[dFirst64KStorage]; -- reset first 64K allocator
    IF pLocation.deviceType = DeviceTypes.ethernet THEN
      BEGIN
      bfn ← pLocation.bootFileNumber;
      SELECT operation FROM
        read => NULL;
        ENDCASE => Error[PilotMP.cGermDeviceError];
      StartRecving[bfn];
      handle ← EFTPGetClump;
      END
    ELSE -- not anything I implement. Pass it on.
```



```
    handle ← RemainingChannels.Create[pLocation, operation, dFirst64KStorage];
    END;

  StartRecving: PROCEDURE [bfn: CARDINAL] =
    BEGIN
    bytes: INTEGER;
    id: PupTypes.Pair;
    type: PupTypes.PupType;
    counter: CARDINAL;
    Timer: PROCEDURE RETURNS [BOOLEAN] =
      BEGIN RETURN[(counter ← counter - 1) = 0]; END;
    iocb ← Allocate[ --SIZE[MiniEthernetDefs.iocb]--16];
    buffer ← LONG[ResidentMemory.AllocateMDS[countTotalBuffer]];
    IF ~MiniEthernetDefs.ActivateDriver[buffer, bufferLength, iocb, TRUE] THEN
      Error[CantActivateDriver];
    him ← [[0], [0], [0, 0]];
    recvStarted ← FALSE;
    receiveSeqNumber ← 0;
    THROUGH [0..15) DO
      MiniEthernetDefs.SendPacket[
        anyBootServer, mySocket, bootFileSend, [0, bfn], NIL, 0];
      counter ← LAST[CARDINAL];
      -- Borrow tail of buffer to discard first packet
      [bytes, id, type] ← MiniEthernetDefs.RecvPacket[
        @him, mySocket, buffer + fudge, bufferLength, Timer];
      IF bytes < 0 THEN LOOP;
      IF type # eData THEN LOOP;
      IF bytes # Environment.bytesPerPage THEN Error[FunnySizePacket];
      -- Discard first packet, it's the Alto boot loader
      SendAck[];
      RETURN;
      REPEAT FINISHED => Error[NoBootServerResponded];
      ENDLOOP;
    END;

  EFTPGetClump: BootChannel.Handle
    --PROCEDURE [page: PageNumber, count: PageCount]-- =
    BEGIN
    where: LONG POINTER ← LPFromPage[page];
    bytes: INTEGER;
    id: PupTypes.Pair;
    type: PupTypes.PupType;
    counter: CARDINAL;
    Timer: PROCEDURE RETURNS [BOOLEAN] =
      BEGIN RETURN[(counter ← counter - 1) = 0] END;
    IF count = 0 THEN BEGIN StopRecving[bfn]; RETURN END;
    -- transfers done. Shutdown the booter.
    THROUGH [0..count) DO
      DO
        -- Handy control structure
        counter ← LAST[CARDINAL];
        [bytes, id, type] ← MiniEthernetDefs.RecvPacket[
          @him, mySocket, where, wordsPerPage, Timer];
        IF bytes < 0 THEN Error[NextPacketDidntArrive];
        IF type # eData THEN LOOP;
        IF bytes # Environment.bytesPerPage THEN Error[FunnySizePacket];
        SELECT id.b FROM
          receiveSeqNumber + 1 => EXIT;
```

```
        receiveSeqNumber =>
          BEGIN -- Booter lost our ack
          SendAck[];
          LOOP;
          END;
        ENDCASE => Error[FunnySequenceNumber];
      ENDLOOP;
    where ← where + wordsPerPage;
    receiveSeqNumber ← receiveSeqNumber + 1;
    SendAck[];
    ENDLOOP;
  END;

  StopRecving: PROCEDURE [bfn: CARDINAL] =
    BEGIN
    bytes: INTEGER;
    id: PupTypes.Pair;
    type: PupTypes.PupType;
    counter: CARDINAL;
    Timer: PROCEDURE RETURNS [BOOLEAN] =
      BEGIN RETURN[(counter ← counter - 1) = 0]; END;
    DO
      counter ← LAST[CARDINAL];
      -- Borrow tail of buffer to process tail
      [bytes, id, type] ← MiniEthernetDefs.RecvPacket[
        @him, mySocket, buffer + fudge, bufferLength, Timer];
      IF bytes < 0 THEN Error[NextPacketDidntArrive];
      SELECT TRUE FROM
        (type = eData AND id.b = receiveSeqNumber) =>
          BEGIN -- Booter lost our ack
          SendAck[];
          LOOP;
          END;
        (type = eData AND id.b = receiveSeqNumber + 1) =>
          BEGIN -- Discard LoadState and BCD at end of file
          receiveSeqNumber ← receiveSeqNumber + 1;
          SendAck[];
          LOOP;
          END;
        (type = eEnd AND id.b = receiveSeqNumber + 1) =>
          BEGIN receiveSeqNumber ← receiveSeqNumber + 1; SendAck[]; EXIT; END;
        ENDCASE => Error[FunnySequenceNumber];
      ENDLOOP;
    MiniEthernetDefs.KillDriver[];
    --Freelocb[iocb];  + + deallocate not implemented.
    --ResidentMemory.FreeMDSPages[base: Inline.LowHalf[buffer], pages: countTotalBuffer];  +
** + deallocate not implemented.

    END;

  SendAck: PROCEDURE =
    BEGIN
    MiniEthernetDefs.SendPacket[
      him, mySocket, eAck, [0, receiveSeqNumber], NIL, 0];
    END;
  -- Allocator for 16-word aligned storage in first64K

  Allocate: PROCEDURE [size: CARDINAL] RETURNS [lp: LONG POINTER TO UNSPECIFIED] =
```



```
    BEGIN pAllocateNext ← (lp ← pAllocateNext) + LONG[((size + 15)/16)*16] END;

  Error: SIGNAL [PilotMP.Code] = CODE;

  LPFromPage: PROCEDURE [page: PageNumber] RETURNS [LONG POINTER] = INLINE
    BEGIN RETURN[LP[highbits: page/256, lowbits: page*256 --MOD 216--]] END;

  END.
```

LOG

Time: February 7, 1980 2:28 PM By: Knutsen and Murray Action: Create file from
***BootChannelDisk of February 6, 1980 2:43 PM*
Buffer unscrambling: February 8, 1980 12:12 AM
Discard Tail: February 8, 1980 1:08 AM
Time: February 8, 1980 12:12 AM By: Knutsen and Murray Action: Buffer unscram
***bling*
Time: February 8, 1980 1:08 AM, By: Knutsen and Murray Action: Discard trailing
***garbage on end of boot file (bcd for the loader?)*
Time: February 8, 1980 10:21 PM By: Forrest Action: Put MP hack to show pag
***e 0*
Time: February 9, 1980 11:11 AM By: Knutsen Action: Loop when page 0 seen s
***econd time*
Time: February 9, 1980 1:34 PM By: Knutsen Action: Fudge in extra space befo
***re buffer*
Time: February 9, 1980 2:39 PM By: Knutsen Action: display packet seq numbe
***r in cursor*
Time: February 9, 1980 1:34 PM By: Knutsen/HGM Action: Undo fudge
Time: February 14, 1980 5:23 PM By: McJones Action: New Boot.Location, faces,
***etc.*
Time: April 17, 1980 10:14 PM By: Luniewski Action: AllocateMDSPages -> AllocateMDS
Time: April 22, 1980 5:37 PM By: McJones Action: Add avoidCleanup: TRUE to Activate
***Driver call*
Time: June 23, 1980 2:37 PM By: McJones Action: OISProcessorFace = >ProcessorFac
***e*

```
-- BootSwapGerm.mesa (last edited by: Forrest on: October 12, 1980 3:39 PM)
-- note: I think the code in initialize assumes the machine starts the germ
-- with memory maped at address 0

DIRECTORY
  Boot USING [
    bootPhysicalVolume, inLoad, Location, MDSIndex, outLoad, pRequest,
    pInitialLink, ReadMDS, teledebug],
  BootChannel USING [Create, Operation, Handle, Transfer],
  BootFile USING [
    Continuation, currentVersion, Entry, Header, InLoadMode, maxEntriesPerHeader,
    maxEntriesPerTrailer, Trailer],
  BootSwap USING [
    countSkip, Initialize, InitializeMDS, mdsiGerm, pCountGerm, pMon,
    ResponseKind],
  Environment USING [Long, maxPagesInMDS, PageCount, PageNumber, wordsPerPage],
  Frame USING [Free, GetReturnFrame, GetReturnLink, SetReturnLink],
  HeadStartChain USING [Start],
  Inline USING [BITAND],
  Mopcodes USING [zRET, zSLB],
  ProcessorFace USING [SetMP, Start],
  PageMap USING [
    Assoc, Flags, flagsClean, flagsVacant, GetF, RealPageNumber, SetF, Value,
    valueVacant],
  PhysicalVolumeFormat USING [currentVersion, Descriptor, Seal],
  PilotMP USING [
    cCantSwap, cGerm, cGermAction, cGermAllocTrap, cGermBadBootFile,
    cGermBadPhysicalVolume, cGermControlFault, cGermDeviceError, cGermDriver,
    cGermERROR, cGermFinished, cGermMemoryFault, cGermStartFault, Code],
  PrincOps USING [
    ControlLink, ControlModule, FrameHandle, GlobalFrameHandle, MainBodyIndex,
    NullGlobalFrame, NullLink, Port, returnOffset, StateVector, UnboundLink],
  PrincOpsRuntime USING [GetFrame, GFT],
  ProcessOperations USING [WriteWDC],
  PSB USING [MonitorLock, UnlockedEmpty],
  ResidentMemory USING [AllocateMDS, AllocateMDSInternal],
  SDDefs USING [
    sAllocTrap, sControlFault, SD, sError, sPageFault, sSignal, sStart, sSwapTrap,
    sWriteProtect],
  Teledebug USING [Debug],
  Trap USING [ReadOTP];

BootSwapGerm: MONITOR LOCKS residentMemoryLock -- LOCKS BootSwap.pMon, --
  IMPORTS
    Boot, BC: BootChannel, BootSwap, Frame, Heads: HeadStartChain, Inline,
    ProcessorFace, PageMap, PrincOpsRuntime, ProcessOperations,
    importedResidentMemory: ResidentMemory, Teledebug, Trap
  -- IMPORTs ResidentMemory so that we can use Pilot's during testing.

  EXPORTS BootChannel, HeadStartChain, ResidentMemory
  SHARES Boot, BootSwap, PageMap, ResidentMemory =
  BEGIN OPEN BootChannel, BootFile, BootSwap, Environment, PageMap, PilotMP;
  -- Do NOT save any status between DoOutLoad and DoInLoad relating to the request.
  -- The next time everything in the outside world may have changed.
  -- The booting action defined by the Principles of Operation should include:
  --  1. Put all usable real memory somewhere in virtual memory.
  --  2. Read countGerm pages of a "boot swap germ" into virtual memory beginning at
  --     page pageGerm + countSkip (steal real memory from high end to do this).
```



```
  --  3. Set Boot.pRequest↑ to [inLoad, locationOfBootFile].
  --  4. Set WDC>0, NWW = 0, MDS = pageGerm, STKP = 0.
  --  5. Xfer[dest: Boot.pInitialLink].

  -- At present, buffer space allocated for the BootChannel's is reclaimed when the request is com
**plete.  Currently only the BootChannelEther allocates buffer space.  When non-initial-booting o
**perations via the ether are implemented, we will have to hang onto this buffer space forever.

  countVM: PageCount = --ProcessorFace.?--40000B;
  pageGerm: PageNumber = GetPageMDS[] + countSkip;
  -- should be a constant engraved in stone (Boot?)
  pageBuffer: PageNumber; -- page used to read/write boot file map.
  pageGermKeep: PageNumber;
  -- highest (mapped) page of germ VM that we retain for all time.
  pageAfterGerm: PageNumber ← pageGerm + pCountGerm↑;
  -- current end of (mapped) germ storage (includes allocated buffer space,
  -- doesn't include pageTemp).
  -- Note that DoInLoad uses a scratch VM page at pageAfterGerm.

  dFirst64KStorage: LONG DESCRIPTOR FOR ARRAY OF WORD = DESCRIPTOR[
    LOOPHOLE[LONG[177200B], LONG POINTER], 200B];
  -- a piece of the first 64K that we know nobody uses, for allocating IOCB's.
  -- When the germ moves to the first 64K, this should be put in its global frame.

  -- Boot: ENTRY PROC =
  --  BEGIN
  --  continuation: Continuation;
  --  ProcessorFace.SetMP[cGerm];
  --  continuation ← DoInLoad[BC.Create[@Boot.pRequest.location]];
  --  pMon.state ← response; + + i.e. not request
  --  WITH continuation SELECT FROM
  --   initial = >
  --    BEGIN
  --    pMon.responseKind ← initiated;
  --    mon.message↑ ← . . . boot parameter . . .;
  --    Set up psb with mds, destination; requeue it
  --    END;
  --   resumptive = >
  --    BEGIN
  --    pMon.responseKind ← resumed;
  --    NOTIFY pMon.condResponse
  --    END;
  --   ENDCASE = >
  --    Error[cGermBadBootFile];
  --  JumpCall0[OutLoadProcess]
  --  END;

  -- OutLoadProcess: INTERNAL PROC =
  --  BEGIN
  --  Condition: TYPE = MACHINE DEPENDENT RECORD [queue, timeout: UNSPECIFIED];
  --  DO
  --   Wait for request and process it
  --   WHILE pMon.state~ = request DO WAIT pMon.condRequest ENDLOOP;
  --   ProcessorFace.SetMP[cGerm];
  --   DoOutLoad[BC.Create[@Boot.pRequest.location], pMon.inLoadMode,
  --    Continuation[
  --     fill: 0,
  --     vp: resumptive[condResponse: LOOPHOLE[pMon.condResponse, Condition].queue]]];
```

```
--  Send response
--  pMon.state ← response;
--  pMon.responseKind ← outLoaded;
--  NOTIFY pMon.condResponse
--  ENDLOOP
--  END;

Run: PROC =  -- after the very first execution, the germ entry point is here
  BEGIN
  handle: BootChannel.Handle;
  ProcessorFace.SetMP[cGermAction];
  BootSwap.InitializeMDS[]; -- set pMon
  DO
    -- inLoad exits through JumpCall2, outLoad does explicit EXIT
    -- bootPhysicalVolume turns itself into an inLoad
    SELECT Boot.pRequest.action FROM
      Boot.inLoad =>
        BEGIN
        continuation: Continuation;
        responseKind: BootSwap.ResponseKind;
        mdsiOther: Boot.MDSIndex;
        destOther: UNSPECIFIED;
        handle ← BC.Create[@Boot.pRequest.location, read, dFirst64KStorage];
        [continuation, pMon.pStartListHeader] ← DoInLoad[handle];
        WITH continuation SELECT FROM
          initial --[mdsi, destination]-- =>
            BEGIN
            responseKind ← initiated;
            mdsiOther ← mdsi;
            destOther ← destination
            END;
          resumptive --[mdsi, resumee]-- =>
            BEGIN
            responseKind ← resumed;
            mdsiOther ← mdsi;
            destOther ← resumee
            END;
          ENDCASE => Error[cGermBadBootFile];
        BootSwap.Initialize[mdsiOther]; -- set (pMon and) WriteMDS machinery
        pMon.responseKind ← responseKind;
        ProcessorFace.SetMP[cGermFinished];
        JumpCall2[arg1: mdsiOther, arg2: destOther, Proc: pMon.CrossMDSCall];
        -- free our frame

        END;
      Boot.outLoad =>
        BEGIN
        handle ← BC.Create[@Boot.pRequest.location, write, dFirst64KStorage];
        DoOutLoad[handle, pMon.inLoadMode, pMon.continuation];
        pMon.responseKind ← outLoaded;
        EXIT;
        END;
      Boot.bootPhysicalVolume =>
        BEGIN
        pvDesc: POINTER TO PhysicalVolumeFormat.Descriptor = PointerFromPage[
          pageBuffer];
        Boot.pRequest.location.diskFileID.da ← [0, 0]; --kludge!
        handle ← BC.Create[@Boot.pRequest.location, rawRead, dFirst64KStorage];
        Transfer[handle, pageBuffer, 1];
```



```
        IF pvDesc.seal ~= PhysicalVolumeFormat.Seal OR pvDesc.version ~=
          PhysicalVolumeFormat.currentVersion THEN
          Error[cGermBadPhysicalVolume];
        Transfer[handle, 0, 0]; -- shut down channel
        Boot.pRequest.location.diskFileID ← pvDesc.bootingInfo[pilot];
        Boot.pRequest.action ← Boot.inLoad;
        -- now get the boot file and go

        END;
      Boot.teledebug =>
        IF IsBound[Teledebug.Debug] THEN {
          Teledebug.Debug[pageBuffer, dFirst64KStorage]; EXIT}
        ELSE Error[cCantSwap];
      ENDCASE => Error[cGermERROR];
    ENDLOOP;
  ProcessorFace.SetMP[cGermFinished];
  END;

DoInLoad: PROC [channel: BootChannel.Handle]
  RETURNS [continuation: Continuation, pStartListHeader: POINTER] =
  BEGIN
  inLoadMode: InLoadMode;
  realPageTemp: RealPageNumber; -- SkipPageHack
  count, countData, countGroup, countRemaining: PageCount;
  page, pageLastMapped, pageNext, pageRun: PageNumber;
  pBuffer: POINTER = PointerFromPage[pageBuffer];
  nEntry: CARDINAL;
  pEntry, pEntryGroup: POINTER TO Entry;
  value: Value;
  SkipPage: PROC = -- SkipPageHack
    BEGIN
    pageTemp: PageNumber = pageAfterGerm;
    Assoc[pageTemp, Value[FALSE, flagsClean, realPageTemp]];
    Transfer[channel, pageTemp, 1];
    Assoc[pageTemp, valueVacant];
    END; --
  -- Read first page, containing header
  Transfer[channel, pageBuffer, 1];
  BEGIN OPEN header: LOOPHOLE[pBuffer, POINTER TO Header];
  IF header.version ~= BootFile.currentVersion THEN Error[cGermBadBootFile];
  pStartListHeader ← header.pStartListHeader;
  SELECT (inLoadMode ← header.inLoadMode) FROM
    load =>
      BEGIN
      CompactVM[];
      pageLastMapped ←
        FIRST[PageNumber] + GetCountRunMapped[FIRST[PageNumber]] - 1;
      realPageTemp ← PageMap.GetF[pageLastMapped].realPage; -- SkipPageHack

      END;
    restore => NULL
    ENDCASE;
  continuation ← header.continuation;
  countData ← header.countData;
  pEntryGroup ← @header.entries[0];
  nEntry ← maxEntriesPerHeader;
  END;
  --
```

```
-- Restore real memory and hardware map from boot file, beginning
-- with data of first group
page ← 0; -- next page to restore
countRemaining ← countData;
DO
  -- Calculate group size
  countGroup ← MIN[nEntry, countRemaining];
  -- Set up map entries to be vacant or nonvacant (and writable), as appropriate
  pEntry ← pEntryGroup;
  THROUGH [0..countGroup) DO
    WHILE page < pEntry.page DO
      -- nongerm pages not in boot file are vacant
      --IF ~InGerm[page] THEN
      IF ~(page IN [pageGerm..pageAfterGerm) OR page = 376B OR page = 377B)
        THEN
        BEGIN
        value ← SetF[page, valueVacant];
        IF inLoadMode = load AND ~IsVacant[value] THEN
          Assoc[pageLastMapped ← pageLastMapped + 1, value]
        END;
      page ← page + 1;
      ENDLOOP;
    IF inLoadMode = restore THEN
      Assoc[page, Value[FALSE, flagsClean, pEntry.value.realPage]];
    pEntry ← pEntry + SIZE[Entry];
    page ← page + 1;
    ENDLOOP;
  -- Transfer data to nonvacant pages
  pEntry ← pEntryGroup;
  count ← countGroup;
  WHILE count ~= 0 DO
    -- find and transfer one run
    pageNext ← pageRun ← pEntry.page;
    IF pageRun = 376B OR pageRun = 377B THEN -- SkipPageHack
      {SkipPage[]; pEntry ← pEntry + SIZE[Entry]; count ← count - 1}
    ELSE
      BEGIN
      DO
        -- until end of run found
        count ← count - 1;
        pageNext ← pageNext + 1;
        pEntry ← pEntry + SIZE[Entry];
        IF count = 0 OR pEntry.page ~= pageNext OR pageNext = 376B OR pageNext
          = 377B --SkipPageHack-- THEN EXIT;
        ENDLOOP;
      Transfer[channel, pageRun, pageNext - pageRun];
      END;
    ENDLOOP; -- Restore map entries
  pEntry ← pEntryGroup;
  THROUGH [0..countGroup) DO
    [] ← SetF[pEntry.page, pEntry.value];
    pEntry ← pEntry + SIZE[Entry]
    ENDLOOP;
  -- Prepare for next group
  countRemaining ← countRemaining - countGroup;
  IF countRemaining = 0 THEN EXIT;
  BEGIN OPEN trailer: LOOPHOLE[pBuffer, POINTER TO Trailer];
  Transfer[channel, pageBuffer, 1];
  IF trailer.version ~= BootFile.currentVersion THEN Error[cGermBadBootFile];
```



```
    pEntryGroup ← @trailer.entries[0];
    nEntry ← maxEntriesPerTrailer;
    END;
    ENDLOOP;
  -- All groups are now loaded
  Transfer[channel, 0, 0]; -- final call to allow cleanup
  SELECT inLoadMode FROM
    load =>
      BEGIN
      -- Recover germ buffer storage: We can't do this when non-initial-booting
      -- operations via the ether are implemented.
      FOR page DECREASING IN [pageGermKeep..pageAfterGerm) DO
        Assoc[pageLastMapped ← pageLastMapped + 1, SetF[page, valueVacant]];
        ENDLOOP;
      pageAfterGerm ← pageGermKeep;
      END;
    restore => -- nongerm pages not in boot file are set vacant:
      WHILE page < countVM DO
        --IF ~InGerm[page] THEN
        IF ~(page IN [pageGerm..pageAfterGerm) OR page = 376B OR page = 377B)
          THEN Assoc[page, valueVacant];
        page ← page + 1;
        ENDLOOP;
    ENDCASE;
  END;

DoOutLoad: PROC [
  channel: BootChannel.Handle, inLoadMode: InLoadMode,
  continuation: Continuation] =
  -- Assumptions: interrupts disabled, all devices quiesced
  BEGIN
  count, countData, countGroup, countRemaining: PageCount;
  page, pageNext, pageRun: PageNumber;
  pBuffer: POINTER = PointerFromPage[pageBuffer];
  nEntry: CARDINAL;
  pEntry, pEntryGroup: POINTER TO Entry; --
  -- Construct header in first map page
  -- Number of pages to save is total nonvacant minus those in germ
  page ← countData ← 0;
  DO
    --IF ~IsVacant[PageMap.GetF[page]] THEN
    IF ~ValueAnd[PageMap.GetF[page], valueVacant] = valueVacant THEN
      countData ← countData + 1;
    IF (page ← page + 1) = countVM THEN EXIT;
    ENDLOOP;
  countData ← countData - (pageAfterGerm - pageGerm) - 2 --i.e. 376B and 377B--
  ;
  BEGIN OPEN header: LOOPHOLE[pBuffer, POINTER TO Header];
  header ←
    [creationDate:, -- fill this when we have a Pilot T.O.D clock?
      pStartListHeader: NIL, inLoadMode: inLoadMode, continuation: continuation,
      countData: countData, entries:];
  pEntryGroup ← @header.entries[0];
  nEntry ← maxEntriesPerHeader;
  END;
  --
  -- Write sequence of (map, data) groups
  page ← 0;
```

```
  countRemaining ← countData;
  DO
    -- Calculate group size
    countGroup ← MIN[nEntry, countRemaining]; -- Write map page
    pEntry ← pEntryGroup;
    THROUGH [0..countGroup) DO
      page ← AdvancePage[page];
      pEntry↑ ← [page, PageMap.GetF[page]];
      pEntry ← pEntry + SIZE[Entry];
      page ← page + 1
      ENDLOOP;
    Transfer[channel, pageBuffer, 1]; -- Write data pages
    pEntry ← pEntryGroup;
    count ← countGroup;
    WHILE count ~= 0 DO
      -- find and transfer one run
      pageNext ← pageRun ← pEntry.page;
      DO
        -- until end of run found
        count ← count - 1;
        pageNext ← pageNext + 1;
        pEntry ← pEntry + SIZE[Entry];
        IF count = 0 OR pEntry.page ~= pageNext THEN EXIT;
        ENDLOOP;
      Transfer[channel, pageRun, pageNext - pageRun];
      ENDLOOP; -- Prepare for next group
    countRemaining ← countRemaining - countGroup;
    IF countRemaining = 0 THEN EXIT;
    BEGIN OPEN trailer: LOOPHOLE[pBuffer, POINTER TO Trailer];
    pEntryGroup ← @trailer.entries[0];
    nEntry ← maxEntriesPerTrailer;
    trailer ← [entries:]; -- set version and any other defaultable fields

    END;
    ENDLOOP;
  Transfer[channel, 0, 0]; -- final call to allow cleanup

  END;

AdvancePage: PROC [page: PageNumber] RETURNS [PageNumber] =
  -- Return first page at or after given page which is not vacant or part of the germ
  BEGIN
  --WHILE IsVacant[PageMap.GetF[page]] OR InGerm[page]
  WHILE PageMap.GetF[page].flags = flagsVacant OR
    (page IN [pageGerm..pageAfterGerm) OR page IN [376B..377B]) DO
    page ← page + 1 ENDLOOP;
  RETURN[page]
  END;

-- Find all real pages currently mapped to virtual pages
--  and map contiguous virtual addresses starting at 0 to them
-- expansions of InGerm[] and IsVacant[] are for speed (compiler thinks
-- is has to fabricate actual booleans
-- Treat Page 1 like an IO page (i.e. never relocate) until the DCB is moved out of it.

CompactVM: PROC =
  BEGIN
  pageDest: PageNumber ← 0;
```



```
  FOR pageSource: PageNumber IN [0..countVM) DO
    value: Value;
    IF pageSource IN [pageGerm..pageAfterGerm) OR pageSource IN [376B..377B]
      THEN LOOP;
    IF pageSource = 1 THEN LOOP;
    --  IF ~IsVacant[value ← SetF[pageSource, valueVacant]] THEN LOOP;
    IF (value ← SetF[pageSource, valueVacant]).flags = flagsVacant THEN LOOP;
    value.flags.writeProtected ← FALSE;
    Assoc[pageDest, value]; -- put back in the map
    WHILE
      ((pageDest ← pageDest + 1) IN [pageGerm..pageAfterGerm) OR pageDest = 1 OR
        pageDest IN [376B..377B]) DO ENDLOOP;
    ENDLOOP;
  END;

-- Backstop Create routine to plug end of chain of BootChannel interfaces.

Create: PUBLIC PROC [
  POINTER TO Boot:Location, BootChannel.Operation, LONG DESCRIPTOR FOR ARRAY OF
  WORD] RETURNS [handle: BootChannel.Handle] = {
  Error[cGermDeviceError]; -- nobody implements this type of device!--};

-- Find length of run of virtual pages all mapped to real pages

GetCountRunMapped: PROC [pageStarting: PageNumber]
  RETURNS [countReal: PageCount] =
  BEGIN
  FOR countReal ← 0, countReal + 1 DO
    IF PageMap.GetF[pageStarting + countReal].flags = flagsVacant THEN EXIT
    ENDLOOP
  END;

GetPageMDS: PROC RETURNS [PageNumber] = {
  RETURN[Boot.ReadMDS[]*maxPagesInMDS]; };

IsVacant: PROC [value: Value] RETURNS [BOOLEAN] = INLINE {
  RETURN[value.flags = flagsVacant]};

-- Virtual memory is from 0 through 0 + countVM[]-1, except pages
-- p for which InGerm[p] = TRUE

-- The following procedure has been MANUALLY expanded inline, so changes
-- to it must be propagated
-- In particular, see code which sets countData in DoOutLoad

InGerm: PROC [page: PageNumber] RETURNS [BOOLEAN] = INLINE
  BEGIN RETURN[page IN [pageGerm..pageAfterGerm) OR page IN [376B..377B]] END;

JumpCall0: PROC [Proc: PROC] = MACHINE CODE
  BEGIN Mopcodes.zSLB, PrincOps.returnOffset; Mopcodes.zRET END;

JumpCall2: PROC [
  arg1, arg2: UNSPECIFIED, Proc: PROC [UNSPECIFIED, UNSPECIFIED]] =
  LOOPHOLE[JumpCall0];

PageFromLongPointer: PROC [p: LONG POINTER TO UNSPECIFIED]
  RETURNS [PageNumber] = {
  OPEN l: LOOPHOLE[p, Long]; RETURN[l.highbits*256 + l.lowbits/256]};
```

```
PageFromPointer: PROC [p: POINTER] RETURNS [page: PageNumber] = {
  RETURN[LOOPHOLE[LOOPHOLE[p, CARDINAL]/wordsPerPage + GetPageMDS[]]]};

PointerFromPage: PROC [page: PageNumber] RETURNS [POINTER] = {
  RETURN[LOOPHOLE[(page - GetPageMDS[])*wordsPerPage]]};

Start: PUBLIC PROC = { -- plug the start chain--};

Transfer: PUBLIC PROC [handle: BC.Handle, page: PageNumber, count: PageCount] =
  -- Only public to make compiler happy
  BEGIN
  ProcessorFace.SetMP[cGermDriver];
  BC.Transfer[handle, page, count];
  ProcessorFace.SetMP[cGermAction];
  END;

ValueAnd: PROC [Value, Value] RETURNS [Value] = LOOPHOLE[Inline.BITAND];

Error: SIGNAL [code: PilotMP.Code] = CODE;

--
-- Simple Mesa runtime
Halt: PROC = INLINE BEGIN DO ENDLOOP END;

AwaitAllocTrap: PORT;
InitializeAllocTrapHandler: PROC =
  BEGIN
  LOOPHOLE[AwaitAllocTrap, PrincOps.Port].dest ← Frame.GetReturnLink[];
  AwaitAllocTrap[]; -- reentered via sAllocTrap
  Frame.SetReturnLink[LOOPHOLE[AwaitAllocTrap, PrincOps.Port].dest];
  -- in case anybody looks
  ProcessorFace.SetMP[cGermAllocTrap];
  Halt[]
  END;

IsBound: PROC [link: UNSPECIFIED] RETURNS [BOOLEAN] = {
  RETURN[link ~= PrincOps.UnboundLink AND link ~= PrincOps.NullLink]};

ControlFaultHandler: PROC = -- entered via sControlFault
  BEGIN
  state: RECORD [a, b: UNSPECIFIED, v: PrincOps.StateVector];
  state.v ← STATE; -- resets stack pointer
  ProcessorFace.SetMP[cGermControlFault];
  Halt[]
  END;

InitializeMonitor: PROC [monitor: LONG POINTER TO MONITORLOCK] = {
  LOOPHOLE[monitor, LONG POINTER TO PSB.MonitorLock]↑ ← PSB.UnlockedEmpty};

-- THIS DOES NOT HANDLE CONTROL MODULES (Single or Multiple)
-- Copy code from Traps when/if needed.
-- entered via sStart KFCB (or call from SwapTrapHandler)

StartModule: PROC [cm: PrincOps.ControlModule] =
  BEGIN OPEN PrincOps;
  control: GlobalFrameHandle;
  state: StateVector;
  state ← STATE;
```



```
  IF ~cm.multiple THEN
    BEGIN OPEN dest: cm.frame;
    IF @dest = NullGlobalFrame OR dest.started OR (control ← dest.global[0]) #
      NullGlobalFrame THEN ERROR Error[cGermStartFault];
    -- IF (control←dest.global[0]) # NullGlobalFrame AND ~control.started THEN
    -- StartModule[[frame[control]]];
    IF ~dest.started THEN
      BEGIN
      state.dest ← ControlLink[
        procedure[gfi: dest.gfi, ep: MainBodyIndex, tag: TRUE]];
      state.source ← Frame.GetReturnFrame[];
      dest.started ← TRUE;
      RETURN WITH state
      END
    ELSE IF state.stkptr # 0 THEN ERROR Error[cGermStartFault];
    END
  ELSE ERROR Error[cGermStartFault];
  END;

SwapTrapHandler: PROC = -- entered via sSwapTrap
  BEGIN OPEN PrincOps, PrincOpsRuntime;
  dest: ControlLink;
  state: StateVector;
  frame: GlobalFrameHandle;
  state ← STATE;
  dest ← Trap.ReadOTP[];
  state.dest ← Frame.GetReturnFrame[];
  state.source ← 0;
  DO
    SELECT TRUE FROM
      dest.proc => {frame ← GetFrame[GFT[dest.gfi]]; EXIT};
      dest.indirect => dest ← dest.link↑;
      ENDCASE -- frame -- => {frame ← dest.frame.accesslink; EXIT};
    ENDLOOP;
  IF ~frame.started THEN StartModule[[frame[frame]]];
  frame.code:out ← FALSE;
  RETURN WITH state
  END;

-- entered via sPageFault or sWriteProtect

MemoryFaultHandler: PROC = {
  ProcessorFace.SetMP[PilotMP.cGermMemoryFault]; Halt[]};

-- entered via sSignal or sError KFCB

SignalHandler: PROC [signal: SIGNAL, code: PilotMP.Code] = {
  ProcessorFace.SetMP[code]; Halt[]};


-----------------------------------------------------------
-- ResidentMemory implementation, for allocating storage in the germ's MDS.
-- AllocateMDS may be called only during module initialization!
-- (The configuration of memory must be static during inLoad and outLoad.
-- Also, spare real memory may not be available after the system is initially loaded)

residentMemoryLock: PUBLIC MONITORLOCK; -- (PRIVATE in interface)
allocateMDSInternal: PUBLIC --INTERNAL--ResidentMemory.AllocateMDSInternal ←
```

```
  [AllocateMDSLocal]; -- (PRIVATE in Interface)

-- Guaranteed not to do an ALLOC from the frame heap.
-- note the page is writable thanks to CompactVM
AllocateMDSLocal: PROC [pages: CARDINAL] RETURNS [p: POINTER TO UNSPECIFIED] =
  BEGIN
  p ← PointerFromPage[pageAfterGerm];
  THROUGH [0..pages) DO
    Assoc[
      pageAfterGerm, SetF[
      page: FIRST[PageNumber] + GetCountRunMapped[FIRST[PageNumber]] - 1,
      flagsNew: valueVacant]];
    -- grab last real page (can only do during an initial boot (inLoadMode = inLoad))
    pageAfterGerm ← pageAfterGerm + 1;
    ENDLOOP;
  END;

-----------------------------------------------------------------

testing: BOOLEAN;
-- TRUE, = > we are running in a test environment on top of Pilot
Initialize: PROC = -- Assume called from main body
  -- Assume all SD entries zero
  BEGIN OPEN SDDefs;
  pSD: POINTER TO ARRAY [0..0) OF UNSPECIFIED ← SD;
  state: PrincOps.StateVector;
  mainbody: PrincOps.FrameHandle = Frame.GetReturnFrame[];
  testing ← pageGerm ~= (mdsiGerm*maxPagesInMDS) + countSkip;
  -- TRUE if not loaded in normal place
  InitializeAllocTrapHandler[];
  pSD[sAllocTrap] ← @AwaitAllocTrap;
  pSD[sControlFault] ← ControlFaultHandler;
  pSD[sSwapTrap] ← SwapTrapHandler;
  IF testing THEN --Pilot is handling traps--NULL
  ELSE pSD[sPageFault] ← pSD[sWriteProtect] ← MemoryFaultHandler;
  pSD[sSignal] ← pSD[sError] ← SignalHandler;
  pSD[sStart] ← StartModule;
  ProcessorFace.Start[];
  InitializeMonitor[@residentMemoryLock];
  --check to see if the page following the germ is already mapped.
  --This may be the case with microcode swapping
  IF PageMap.GetF[pageAfterGerm].flags # PageMap.flagsVacant THEN
    pageAfterGerm ← (pageBuffer ← pageAfterGerm) + 1
  ELSE
    pageBuffer ← PageFromPointer[importedResidentMemory.AllocateMDS[pages: 1]];
  pageGermKeep ← pageAfterGerm;
  -- we keep the buffer page forever, deallocate all else after each request.
  -- We can't deallocate when non-initial-booting operations via the
  -- ether are implemented.
  Heads.Start[]; -- start the heads (face implementations)
  state.instbyte ← state.stkptr ← 0;
  state.dest ← Boot.pInitialLink↑ ← LOOPHOLE[Run, PrincOps.ControlLink];
  -- subsequent entries to the germ go directly to Run[]
  state.source ← mainbody.returnlink;
  Frame.Free[mainbody];
  RETURN WITH state -- to Run[], never to return to Initialize

  END;
```



```
  ProcessorFace.SetMP[cGerm]; -- let them know we are here
  ProcessOperations.WriteWDC[1]; -- temporary hack for Dandelion...delete soon
  Initialize[]; -- never returns

  END.
```

LOG
For earlier log entries see Amargosa archive version.
Time: April 10, 1980 11:59 AM By: Forrest Action: Implement call to Teledebug.Debug.
Time: April 17, 1980 12:44 AM By: Forrest Action: Get teledebug from Boot not Bootsw**ap
Time: April 17, 1980 10:39 PM By: Luniewski Action: AllocateMDSPages = >AllocateMDS
Time: April 18, 1980 3:58 PM By: Knutsen Action: Export allocateMDSInternal, resident**MemoryLock
Time: May 15, 1980 6:18 PM By: McJones Action: Mesa 6; call OISProcessorFace.Start**; temporarily initialize WDC
Time: June 10, 1980 9:16 AM By: Forrest Action: PrincOps traps/use physical volume**Format
Time: June 23, 1980 2:42 PM By: McJones Action: OISProcessorFace = >ProcessorFac**e
Time: July 20, 1980 9:18 PM By: Forrest Action: PrincOpsRuntime
Time: August 5, 1980 9:17 AM By: Forrest Action: Add test for Page after germ being m**apped, use GetFlags
Time: September 9, 1980 5:50 PM By: Forrest/McJones Action: Add cGermFinished

```
-- MiniEthernetDriver.mesa (last edited by: Forrest on: September 17, 1980 7:36 PM)

DIRECTORY
  Inline USING [LongCOPY],
  EthernetOneFace,
  BufferDefs,
  DriverTypes USING [ethernetEncapsulationOffset],
  PupTypes,
  MiniEthernetDefs;

MiniEthernetDriver: PROGRAM
  IMPORTS Inline, EthernetOneFace EXPORTS MiniEthernetDefs SHARES BufferDefs =
  BEGIN OPEN EthernetOneFace;

  active: BOOLEAN ← FALSE;
  last: {in, out, reset};
  knowBoardLocation: BOOLEAN ← FALSE;

  myHost: PupTypes.PupHostID;
  myNet: PupTypes.PupNetID;

  b: BufferDefs.PupBuffer; -- Global buffer for actual data transfers
  bufferSize: CARDINAL;
  ether: DeviceHandle;
  longlocb: ControlBlock;
  global: GlobalStatePtr;

  wordsPerPupHeader: CARDINAL = 11;
  bytesPerPupHeader: CARDINAL = wordsPerPupHeader*2;

  BoardLocationUnknown: PUBLIC ERROR = CODE;
  DriverNotActive: PUBLIC ERROR = CODE;
  BufferOverflow: PUBLIC ERROR = CODE;
  ReturnToStrangePlace: PUBLIC ERROR = CODE;

  ActivateDriver: PUBLIC PROCEDURE [
    dataBuffer: LONG POINTER, length: CARDINAL, iocb: LONG POINTER,
    avoidCleanup: BOOLEAN ← FALSE] RETURNS [BOOLEAN] =
    BEGIN
    p: LONG POINTER ← @b.encapsulation + DriverTypes.ethernetEncapsulationOffset;
    q: LONG POINTER ← b;
    fudge: LONG CARDINAL ← p - q;
    net, host: CARDINAL;
    IF ~FindTheBoard[] THEN RETURN[FALSE];
    knowBoardLocation ← TRUE;
    b ← dataBuffer - fudge;
    bufferSize ← length;
    longlocb ← iocb;
    [net, host] ← GetEthernet1Address[ether];
    myHost ← [host];
    myNet ← [net];
    IF ~avoidCleanup THEN AddCleanup[ether];
    -- allocate global storage
    TurnOn[ether, myHost, 0, 0, global];
    active ← TRUE;
    last ← reset;
    RETURN[TRUE];
    END;
```



```
  KillDriver: PUBLIC PROCEDURE [avoidCleanup: BOOLEAN ← FALSE] =
    BEGIN
    TurnOff[ether];
    IF ~avoidCleanup THEN RemoveCleanup[ether];
    knowBoardLocation ← FALSE;
    END;

  GetEthernetHostNumber: PUBLIC PROCEDURE RETURNS [CARDINAL] =
    BEGIN
    net, host: CARDINAL;
    IF NOT knowBoardLocation THEN ERROR BoardLocationUnknown;
    [net, host] ← GetEthernet1Address[ether];
    RETURN[host];
    END;

  GetEthernetNetNumber: PUBLIC PROCEDURE RETURNS [CARDINAL] =
    BEGIN
    net, host: CARDINAL;
    IF NOT knowBoardLocation THEN ERROR BoardLocationUnknown;
    [net, host] ← GetEthernet1Address[ether];
    RETURN[IF active THEN myNet ELSE net];
    END;

  FindTheBoard: PROCEDURE RETURNS [BOOLEAN] =
    BEGIN
    ether ← GetNextDevice[nullDeviceHandle];
    RETURN[ether # nullDeviceHandle];
    END;

  SendPacket: PUBLIC PROCEDURE [
    dest: PupTypes.PupAddress, me: PupTypes.PupSocketID, type: PupTypes.PupType,
    id: PupTypes.Pair, data: LONG POINTER, bytes: CARDINAL] =
    BEGIN
    words: CARDINAL ← 2 + wordsPerPupHeader + (bytes + 1)/2;
    IF ~active THEN ERROR DriverNotActive;
    IF words > bufferSize THEN ERROR BufferOverflow;
    -- Don't use constructor since that clobbers preceding 4 words
    b.encapsulation.etherDest ← dest.host;
    b.encapsulation.etherSource ← myHost;
    b.encapsulation.ethernetType ← pupEthernetPacket;
    b.pupLength ← bytesPerPupHeader + bytes;
    b.pupTransportControl ← 0;
    b.pupType ← type;
    b.pupID ← id;
    b.dest ← dest;
    b.source ← [myNet, myHost, me];
    Inline.LongCOPY[to: @b.pupWords, from: data, nwords: (bytes + 1)/2];
    SetPupChecksum[b];
    last ← out;
    QueueOutput[
      ether, @b.encapsulation + DriverTypes.ethernetEncapsulationOffset, words,
      longlocb];
    THROUGH [0..LAST[CARDINAL]) DO
      IF GetStatus[longlocb] # pending THEN EXIT;
      REPEAT
        FINISHED =>
          BEGIN TurnOff[ether]; TurnOn[ether, myHost, 0, 0, global]; END;
      ENDLOOP;
    END;
```

```
ReturnPacket: PUBLIC PROCEDURE [
  type: PupTypes.PupType, data: LONG POINTER, bytes: CARDINAL] =
  BEGIN
  words: CARDINAL ← 2 + wordsPerPupHeader + (bytes + 1)/2;
  temp: PupTypes.PupAddress;
  IF ~active THEN ERROR DriverNotActive;
  IF words > bufferSize THEN ERROR BufferOverflow;
  IF last # in THEN ERROR ReturnToStrangePlace;
  -- Don't use constructor since that clobbers preceding 4 words
  b.encapsulation.etherDest ← b.encapsulation.etherSource;
  b.encapsulation.etherSource ← myHost;
  b.encapsulation.ethernetType ← pupEthernetPacket;
  b.pupLength ← bytesPerPupHeader + bytes;
  b.pupTransportControl ← 0;
  b.pupType ← type;
  temp ← b.dest;
  b.dest ← b.source;
  b.source ← temp;
  Inline.LongCOPY[to: @b.pupWords, from: data, nwords: (bytes + 1)/2];
  SetPupChecksum[b];
  last ← out;
  QueueOutput[
    ether, @b.encapsulation + DriverTypes.ethernetEncapsulationOffset, words,
    longlocb];
  THROUGH [0..LAST[CARDINAL]) DO
    IF GetStatus[longlocb] # pending THEN EXIT;
    REPEAT
      FINISHED =>
        BEGIN TurnOff[ether]; TurnOn[ether, myHost, 0, 0, global]; END;
    ENDLOOP;
  END;

RecvPacket: PUBLIC PROCEDURE [
  source: LONG POINTER TO PupTypes.PupAddress, me: PupTypes.PupSocketID,
  data: LONG POINTER, words: CARDINAL, timeout: PROCEDURE RETURNS [BOOLEAN]]
  RETURNS [bytes: CARDINAL, id: PupTypes.Pair, type: PupTypes.PupType] =
  BEGIN
  IF ~active THEN ERROR DriverNotActive;
  DO
    QueueInput[
      ether, @b.encapsulation + DriverTypes.ethernetEncapsulationOffset,
      bufferSize, longlocb];
    UNTIL GetStatus[longlocb] # pending DO
      IF timeout[] THEN.
        BEGIN
        TurnOff[ether];
        TurnOn[ether, myHost, 0, 0, global];
        last ← reset;
        RETURN[MiniEthernetDefs.timedOut, [0, 0], LOOPHOLE[0]];
        END;
      ENDLOOP;
    IF GetStatus[longlocb] # ok THEN LOOP;
    IF b.encapsulation.ethernetType # pupEthernetPacket THEN LOOP;
    -- should check words that arrived against pupLength?
    -- check here for routing table if we ever get that complicated
    IF myNet # 0 AND myNet # b.dest.net AND b.dest.net # 0 THEN LOOP;
    IF myHost # b.dest.host THEN LOOP; -- Can't recv broadcast
    IF me # b.dest.socket THEN LOOP;
    IF source.net # 0 AND source.net # b.source.net AND b.source.net # 0 THEN
```



```
      LOOP;
    IF source.host # 0 AND source.host # b.source.host THEN LOOP;
    IF source.socket # [0, 0] AND source.socket # b.source.socket THEN LOOP;
    IF ~TestPupChecksum[b] THEN LOOP;
    IF myNet = 0 THEN myNet ← b.dest.net;
    IF (b.pupLength - bytesPerPupHeader) > words*2 THEN LOOP;
    bytes ← b.pupLength - bytesPerPupHeader;
    Inline.LongCOPY[to: data, from: @b.pupWords, nwords: (bytes + 1)/2];
    IF source.net = 0 THEN source.net ← b.source.net;
    IF source.host = 0 THEN source.host ← b.source.host;
    IF source.socket = [0, 0] THEN source.socket ← b.source.socket;
    last ← in;
    RETURN[bytes, b.pupID, b.pupType];
    ENDLOOP;
  END;

checksum: {software} = software;

SetPupChecksum: PUBLIC PROCEDURE [b: BufferDefs.PupBuffer] =
  BEGIN
  size: CARDINAL ← (b.pupLength - 1)/2;
  checksumLoc: LONG POINTER ← @b.pupLength + size;
  cs, t: CARDINAL;
  SELECT checksum FROM
    software =>
      BEGIN
      p: LONG POINTER TO ARRAY [0..0) OF WORD = LOOPHOLE[@b.pupLength];
      i: CARDINAL;
      cs ← 0;
      FOR i IN [0..size) DO
        t ← cs + p[i];
        cs ← (IF cs > t THEN t + 1 ELSE t);
        cs ← (IF cs >= 100000B THEN cs*2 + 1 ELSE cs*2);
        ENDLOOP;
      IF cs = 177777B THEN cs ← 0;
      checksumLoc↑ ← cs;
      END;
    ENDCASE => NULL;
  END;

TestPupChecksum: PUBLIC PROCEDURE [b: BufferDefs.PupBuffer] RETURNS [BOOLEAN] =
  BEGIN
  size: CARDINAL ← ((LOOPHOLE[b.pupLength - 1, CARDINAL])/2);
  checksumLoc: LONG POINTER ← @b.pupLength + size;
  cs, t: CARDINAL;
  IF checksumLoc↑ = 177777B THEN RETURN[TRUE];
  SELECT checksum FROM
    software =>
      BEGIN
      p: LONG POINTER TO ARRAY [0..0) OF WORD = LOOPHOLE[@b.pupLength];
      i: CARDINAL;
      cs ← 0;
      FOR i IN [0..size) DO
        t ← cs + p[i];
        cs ← (IF cs > t THEN t + 1 ELSE t);
        cs ← (IF cs >= 100000B THEN cs*2 + 1 ELSE cs*2);
        ENDLOOP;
      IF cs = 177777B THEN cs ← 0;
```

```
        END;
      ENDCASE => NULL;
    RETURN[checksumLoc↑ = cs];
    END;

  END.
```

LOG
Time: January 29, 1980 6:00 PM By: Dalal Action: fixed Core Software AR 2658.
Time: February 7, 1980 11:18 PM By: Knutsen Action: ioPage now a long pointer. Initialize host s
***ocket right.*
INTEGER Divide
Time: February 9, 1980 2:43 PM By: Knutsen/HGM Action: fix clobber of 4 words before buffer
Time: April 20, 1980 4:03 PM By: HGM Action: Faceification, add ReturnPacket and timedOut
Time: April 28, 1980 7:36 PM By: HGM Action: Fix QueueOutput bug (again!) in RecvPacket
Time: July 2, 1980 5:44 PM By: HGM Action: Delete "DriverTypes." in front of pupEthernetPacket
Time: August 20, 1980 2:23 PM By: BLyon Action: renamed EthernetFace to EthernetOneFace.

```
-- TeledebugImpl.mesa (last edited by: Forrest September 17, 1980 2:27 PM)
-- This is an initial, ugly, ugly, implementation.

DIRECTORY
  DeviceTypes USING [sa4000],
  Environment USING [PageNumber, wordsPerPage],
  Inline USING [BITAND, LongCOPY, LongDivMod, LowHalf],
  MiniEthernetDefs USING [ActivateDriver, KillDriver, RecvPacket, ReturnPacket],
  PageMap USING [Assoc, GetF, Value, valueVacant],
  PilotDisk USING [Label],
  PilotMP USING [cGermDeviceError, Code, cTeledebugServer],
  ProcessorFace USING [SetMP],
  PupTypes,
  TeledebugProtocol,
  SA4000Face,
  Teledebug,
  Utilities USING [LongPointerFromPage];

TeledebugImpl: PROGRAM
  IMPORTS Inline, MiniEthernetDefs, PageMap, ProcessorFace, SA4000Face, Utilities
  EXPORTS Teledebug
  SHARES PageMap =
  BEGIN

  -- Ethernet buffer. This is merely storage for MiniDriver
  bufSize: CARDINAL = 300;
  fudge: CARDINAL = 30;
  eBuff: ARRAY [0..bufSize + fudge + 3) OF UNSPECIFIED;
  eBuffer: POINTER = Inline.BITAND[@eBuff + 3, 177774B];
  etherIOCBSize: CARDINAL = 30;

  Error: SIGNAL [PilotMP.Code] = CODE;

  -- "Locals" to Debug and RunDisk
  MyPup: TYPE = RECORD [
    SELECT OVERLAID * FROM
      a => [coreStoreRequest: TeledebugProtocol.CoreStoreRequest],
      b => [coreStoreAcknowledgement: TeledebugProtocol.CoreStoreAcknowledgement],
      c => [coreFetchRequest: TeledebugProtocol.CoreFetchRequest],
      d => [coreFetchAcknowledgement: TeledebugProtocol.CoreFetchAcknowledgement],
      e => [diskStoreRequest: TeledebugProtocol.DiskStoreRequest],
      f => [diskStoreAcknowledgement: TeledebugProtocol.DiskStoreAcknowledgement],
      g => [diskFetchAcknowledgement: TeledebugProtocol.DiskFetchAcknowledgement],
      h => [diskAddressSetRequest: TeledebugProtocol.DiskAddressSetRequest],
      i => [
        diskAddressSetAcknowledgement:
          TeledebugProtocol.DiskAddressSetAcknowledgement],
    ENDCASE];

  diskAddressPup: TeledebugProtocol.DiskAddressSetRequest;
  bytes: INTEGER;
  id: PupTypes.Pair;
  pAllocateNext: LONG POINTER TO UNSPECIFIED; -- iocb's and stuff in first 64K
  pup: MyPup;
  sa4000IOCB: LONG POINTER TO SA4000Face.Operation;
  sender: PupTypes.PupAddress;
  temp: CARDINAL;
  type: PupTypes.PupType;
```



```
  Debug: PUBLIC PROC [
    scratchPage: Environment.PageNumber,
    dFirst64KStorage: LONG DESCRIPTOR FOR ARRAY OF WORD] =
    BEGIN
    Allocate: PROC [size: CARDINAL] RETURNS [lp: LONG POINTER TO UNSPECIFIED] = {
      pAllocateNext ← (lp ← pAllocateNext) + LONG[((size + 15)/16)*16]};
    LongTime: PROC RETURNS [BOOLEAN] = {RETURN[FALSE]};
    SetNoLabel: PROC [p: LONG POINTER] =
      BEGIN OPEN Inline;
      p↑ ← -1;
      LongCOPY[from: p, to: p + 1, nwords: SIZE[TeledebugProtocol.Label] - 1];
      END;

    ProcessorFace.SetMP[PilotMP.cTeledebugServer];
    pAllocateNext ← BASE[dFirst64KStorage]; -- reset first 64K allocator
    SA4000Face.Initialize[
      0, Inline.LowHalf[Allocate[SA4000Face.globalStateSize]]];
    sa4000IOCB ← Allocate[SA4000Face.operationSize];
    IF ~MiniEthernetDefs.ActivateDriver[
      eBuffer, bufSize, Allocate[etherIOCBSize], TRUE] THEN
      Error[PilotMP.cGermDeviceError];
    DO
      sender ← [[0], [0], [0, 0]]; -- talk to anybody
      [bytes, id, type] ← MiniEthernetDefs.RecvPacket[
        @sender, TeledebugProtocol.teleSwatSocket, @pup, SIZE[MyPup], LongTime];
      SELECT type FROM
        TeledebugProtocol.coreFetchRequest =>
          BEGIN OPEN p: pup.coreFetchAcknowledgement;
          IF bytes # (2*SIZE[TeledebugProtocol.CoreFetchRequest]) THEN LOOP;
          IF ~IsVacant[p.page] THEN
            BEGIN
            p.flags ← LOOPHOLE[GetF[p.page]];
            Inline.LongCOPY[
              from: LPFromPage[p.page], to: @p.data,
              nwords: Environment.wordsPerPage];
            Assoc[p.page, LOOPHOLE[p.flags]];
            END
          ELSE p.flags ← TeledebugProtocol.vacantFlag;
          bytes ← 2*TeledebugProtocol.coreFetchAcknowledgementSize;
          END;
        TeledebugProtocol.coreStoreRequest =>
          BEGIN OPEN p: pup.coreStoreRequest;
          IF bytes # (2*SIZE[TeledebugProtocol.CoreStoreRequest]) THEN LOOP;
          IF ~IsVacant[p.page] THEN
            BEGIN
            MakeWritable[p.page];
            Inline.LongCOPY[
              to: LPFromPage[p.page], from: @p.data,
              nwords: Environment.wordsPerPage];
            Assoc[p.page, LOOPHOLE[p.flags]];
            END
          ELSE p.flags ← TeledebugProtocol.vacantFlag;
          bytes ← 2*TeledebugProtocol.coreStoreAcknowledgementSize;
          END;
        TeledebugProtocol.diskAddressSetRequest =>
          BEGIN
          IF bytes # (2*SIZE[TeledebugProtocol.DiskAddressSetRequest]) THEN LOOP;
          Inline.LongCOPY[
            -- just save it.
```

```
        to: @diskAddressPup, from: @pup.diskAddressSetRequest,
        nwords: SIZE[TeledebugProtocol.DiskAddressSetRequest]];
      END;
    TeledebugProtocol.diskFetchRequest =>
      BEGIN OPEN p: pup.diskFetchAcknowledgement;
      SELECT bytes FROM
        0 => NULL;
        2*SIZE[TeledebugProtocol.DiskAddressSetRequest] =>
          Inline.LongCOPY[
            -- save disk address request.
            to: @diskAddressPup, from: @pup.diskAddressSetRequest,
            nwords: SIZE[TeledebugProtocol.DiskAddressSetRequest]];
        ENDCASE => LOOP; -- illegal request
      IF ~RunDisk[vrr, scratchPage, @p.label, @p.data] THEN
        SetNoLabel[@p.label];
      bytes ← 2*TeledebugProtocol.diskFetchAcknowledgementSize;
      END;
    TeledebugProtocol.diskStoreRequest =>
      BEGIN OPEN p: pup.diskStoreRequest;
      IF bytes # (2*SIZE[TeledebugProtocol.DiskStoreRequest]) THEN LOOP;
      IF ~RunDisk[vvw, scratchPage, @p.label, @p.data] THEN
        SetNoLabel[@pup.diskStoreAcknowledgement.label];
      bytes ← 2*TeledebugProtocol.diskStoreAcknowledgementSize;
      END;
    TeledebugProtocol.go =>
      BEGIN
      GoConfirm: PROC RETURNS [BOOLEAN] = INLINE
        BEGIN
        tenSecondsCount: CARDINAL = LAST[CARDINAL];
        MiniEthernetDefs.ReturnPacket[
          TeledebugProtocol.acknowledgement, @pup, 0];
        temp ← tenSecondsCount;
        [bytes, id, type] ← MiniEthernetDefs.RecvPacket[
          @sender, TeledebugProtocol.teleSwatSocket, @pup, SIZE[MyPup],
          TenSeconds];
        -- also check id??? (what to do if wrong)
        RETURN[bytes = -1 OR type = TeledebugProtocol.goReply];
        END;
      TenSeconds: PROC RETURNS [BOOLEAN] = {RETURN[(temp ← temp - 1) = 0]};
      IF bytes = 0 AND GoConfirm[] THEN {
        MiniEthernetDefs.KillDriver[]; RETURN}
      ELSE LOOP; -- unexpected respose; ignore

      END;
    ENDCASE => LOOP;
  -- Send ack
  MiniEthernetDefs.ReturnPacket[
    TeledebugProtocol.acknowledgement, @pup, bytes];
  ENDLOOP;
END;

RunDisk: PROC [
  c: SA4000Face.Command, scratchPage: Environment.PageNumber,
  labelP, dataP: POINTER] RETURNS [BOOLEAN] =
BEGIN OPEN o: sa4000IOCB, SA4000Face;
IF diskAddressPup.device # DeviceTypes.sa4000 THEN RETURN[FALSE];
o.device ← nullDeviceHandle;
THROUGH [0..diskAddressPup.deviceOrdinal] DO
```



```
  IF (o.device ← GetNextDevice[o.device]) = nullDeviceHandle THEN
    RETURN[FALSE];
  ENDLOOP;
[quotient: temp, remainder: o.clientHeader.sector] ← Inline.LongDivMod[
  num: diskAddressPup.page,
  den: GetDeviceAttributes[o.device].sectorsPerTrack];
o.clientHeader.head ← temp MOD GetDeviceAttributes[o.device].movingHeads;
o.clientHeader.cylinder ← temp/GetDeviceAttributes[o.device].movingHeads;
o.labelPtr ← labelP;
o.dataPtr ← LPFromPage[LONG[scratchPage]];
o.incrementDataPtr ← FALSE;
o.command ← c;
o.pageCount ← 1;
IF c = vvw THEN
  Inline.LongCOPY[
    from: LONG[dataP], to: o.dataPtr, nwords: Environment.wordsPerPage];
THROUGH [0..8) DO
  SA4000Face.Initiate[sa4000IOCB];
  DO
    SELECT SA4000Face.Poll[sa4000IOCB] FROM
      inProgress => LOOP;
      goodCompletion =>
        BEGIN
        IF c = vrr THEN
          Inline.LongCOPY[
            to: LONG[dataP], from: o.dataPtr,
            nwords: Environment.wordsPerPage];
        RETURN[TRUE];
        END;
      ENDCASE --ERROR-- => EXIT;
    ENDLOOP;
  ENDLOOP;
RETURN[FALSE];
END;

-- Long page number operations

maxVirtualPages: LONG CARDINAL = 77777B; -- de-burn this constant someday

Assoc: PROC [page: LONG CARDINAL, value: PageMap.Value] = INLINE {
  PageMap.Assoc[Inline.LowHalf[page], value]};

GetF: PROC [page: LONG CARDINAL] RETURNS [PageMap.Value] = INLINE {
  RETURN[PageMap.GetF[Inline.LowHalf[page]]]};

IsVacant: PROC [page: LONG CARDINAL] RETURNS [BOOLEAN] =
  BEGIN OPEN Inline;
  IF page > maxVirtualPages THEN RETURN[FALSE];
  RETURN[BITAND[GetF[page], PageMap.valueVacant] = PageMap.valueVacant];
  END;

LPFromPage: PROC [p: LONG CARDINAL] RETURNS [LONG POINTER] = INLINE {
  RETURN[Utilities.LongPointerFromPage[Inline.LowHalf[p]]]};

MakeWritable: PROC [page: LONG CARDINAL] = {
  v: PageMap.Value ← GetF[page];
  v.flags.writeProtected ← FALSE;
  Assoc[page, v]};
```

```
END.
```

LOG
Time: March 21, 1980 6:58 PM By: Forrest Action: Create file
Time: March 24, 1980 11:45 AMBy: Forrest Action: initialize sender, fix maxVirtualPages
Time: April 1, 1980 2:53 PM By: Forrest Action: Add Disk Implementation
Time: April 7, 1980 7:11 PM By: Forrest Action: Added SetDiskAddress + Fetch enha
***ncement*
Time: April 10, 1980 12:19 PM By: Forrest Action: RCPProtocol = > TeledebugProtocol
Time: April 22, 1980 5:41 PM By: McJones Action: Add avoidCleanup: TRUE to Activate
***Driver call*
Time: June 25, 1980 4:21 PM By: McJones Action: OISDisk = >PilotDisk; OISProcessorF
***ace = >ProcessorFace; SA4000Face.Operation: clientLabel + diskLabel = >labelPtr*
Time: September 17, 1980 12:45 PM By: Forrest Action: change to use returnPack
*.**et; zero socket each trip around loop; use GetF, Utilities.LongPointerFromPage*

```
-- TTYPortDriverA.mesa
-- Last edited: September 30, 1980 10:16 AM By: Artibee

DIRECTORY
  Heap: FROM "Heap" USING [systemZone],
  Process: FROM "Process" USING [MsecToTicks, SetTimeout],
  ProcessInternal: FROM "ProcessInternal" USING [
    AllocateNakedCondition, DeallocateNakedCondition],
  TTYPort: FROM "TTYPort" USING [ChannelHandle],
  TTYPortFace: FROM "TTYPortFace" USING [GetLineCount, Off, On],
  TTYPortInternal: FROM "TTYPortInternal" USING [
    ChannelStatus, ChannelStatusHandle, DeleteChannel, InitializeInterrupts,
    ParameterRecord];

TTYPortDriverA: MONITOR
  IMPORTS Heap, Process, ProcessInternal, TTYPortFace, TTYPortInternal
  EXPORTS TTYPort, TTYPortInternal
  SHARES TTYPort =
  BEGIN
  ChannelAlreadyExists: PUBLIC ERROR = CODE;
  InvalidLineNumber: PUBLIC ERROR = CODE;
  NoTTYPortHardware: PUBLIC ERROR = CODE;
  channelIsCreated: PUBLIC PACKED ARRAY [0..15] OF BOOLEAN ← ALL[FALSE];
  heap: UNCOUNTED ZONE ← Heap.systemZone;
  lineCount: CARDINAL;
  Create: PUBLIC ENTRY PROCEDURE [lineNumber: CARDINAL]
    RETURNS [channel: TTYPort.ChannelHandle] =
    BEGIN -- Extra nesting level so channelStatusHandle defined in UNWIND
    ch: TTYPortInternal.ChannelStatusHandle ← NIL;
    ttyPortDefaults: TTYPortInternal.ParameterRecord =
      [characterLength: lengthIs8bits, clearToSend: FALSE, dataSetReady: FALSE,
        lineSpeed: bps1200, parity: none, stopBits: two];
    BEGIN
    ENABLE UNWIND => BEGIN IF ch # NIL THEN heap.FREE[@ch]; END;
    -- To release monitor lock, return block
    printerWakeupMask: WORD;
    IF lineCount = 0 THEN ERROR NoTTYPortHardware;
    IF lineCount - 1 < lineNumber THEN ERROR InvalidLineNumber;
    IF channelIsCreated[lineNumber] THEN ERROR ChannelAlreadyExists;
    -- allocate and initialize a ChannelStatus record
    ch ← heap.NEW[
      TTYPortInternal.ChannelStatus ←
        [deleteInProgress: FALSE, breakBeingProcessed: FALSE, myDataLost: FALSE,
          lineNumber: lineNumber, parameterRecord: ttyPortDefaults,
          deviceStatus: [FALSE, FALSE, FALSE, FALSE, FALSE, FALSE],
          bufferInput: [timeout: Process.MsecToTicks[10000]], breaksInBuffer: 0,
          charsInBuffer: 0, topInUse: 0, topUnused: 0, buffer: -- buffer --,
          interrupt: NIL, -- process
          wakeUpPtr: NIL, statusChange: [timeout: Process.MsecToTicks[10000]],
          statusWaitCount: 0]];
    [cv: ch.wakeUpPtr, mask: printerWakeupMask] ←
      ProcessInternal.AllocateNakedCondition[];
    Process.SetTimeout[ch.wakeUpPtr, Process.MsecToTicks[10000]];
    TTYPortInternal.InitializeInterrupts[ch];
    -- detach an InterruptProcess to WAIT on printerWakeUp
    TTYPortFace.On[lineNumber, printerWakeupMask];
    channelIsCreated[lineNumber] ← TRUE;
    RETURN[ch];
```



```
    END;
    END;

  Delete: PUBLIC ENTRY PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    ENABLE UNWIND => BEGIN IF channel # NIL THEN heap.FREE[@channel]; END;
    -- To release monitor lock, return block
    TTYPortInternal.DeleteChannel[channel];
    -- modify ChannelStatus record under monitor
    channelIsCreated[channel.lineNumber] ← FALSE;
    TTYPortFace.Off[channel.lineNumber];
    ProcessInternal.DeallocateNakedCondition[channel.wakeUpPtr];
    heap.FREE[@channel];
    -- release block, cannot be in DeleteChannel since LOCK must exist at exit

    END;
  -- MAIN PROGRAM

  lineCount ← TTYPortFace.GetLineCount[];
  IF lineCount > 16 THEN ERROR; -- some tables are hard-coded to max size of 16

  END. -- TTYPortDriverA
```

LOG

*Time: July 18, 1980 4:21 PM By: Mary Artibee Action: Use TTYPortFace; changes to Param**eter, DeviceStatus, TransferStatus.*

*Time: July 28, 1980 6:35 PM By: Mary Artibee Action: Split into 2 driver monitors; this is the**monitor for Channel creation/deletion.*

*Time: July 30, 1980 1:39 AM By: Mary Artibee Action: Added call to InitializeInterrupts; init**bufferInput condition; set NIL nakedNotifyPtr at Delete.*

*Time: August 4, 1980 1:45 PM By: Mary Artibee Action: DeallocateNakedNotifyLevel at Delet**e.*

*Time: August 5, 1980 2:37 PM By: Mary Artibee Action: Use MsecToTicks; InitializeCondition**on printerWakeUp; use ResidentHeap for naked notify condition.*

*Time: August 6, 1980 4:28 PM By: Mary Artibee Action: Moved ttyPortDefaults here; charact**erLength now lengthIs8bits.*

*Time: August 16, 1980 6:02 PM By: Fay Action: Changed spelling: systemZONE => systemZon**e.*

*Time: September 3, 1980 12:14 AM By: Artibee Action: Use ProcessInternal.(De)**AllocateNakedCondition.*

*Time: September 15, 1980 3:03 PM By: Artibee Action: Do DeallocateNakedCond**ition before FREE.*

*Time: September 25, 1980 9:10 AM By: Artibee Action: Changed ch Monitored Re**cord init.*

*Time: September 30, 1980 10:16 AM By: Artibee Action: reflect ChannelStatus cha**nges.*

```
-- TTYPortDriverB.mesa
-- Last edited: October 6, 1980 4:37 PM By: Mary Artibee

DIRECTORY
  Process: FROM "Process" USING [MsecToTicks, Priority, SetPriority, SetTimeout],
  TTYPort: FROM "TTYPort" USING [
    CharacterLength, DeviceStatus, LineSpeed, Parameter, Parity, StopBits,
    TransferStatus],
  TTYPortFace: FROM "TTYPortFace" USING [
    DeviceStatus, GetCommand, GetStatus, Parameter, PutCommand, SetParameter,
    TransferStatus],
  TTYPortInternal: FROM "TTYPortInternal" USING [
    ChannelStatus, ChannelStatusHandle, maxBufferSize];

TTYPortDriverB: MONITOR LOCKS channel USING channel:
  TTYPortInternal.ChannelStatusHandle
  IMPORTS Process, TTYPortFace EXPORTS TTYPort, TTYPortInternal SHARES TTYPort =
  BEGIN
  ChannelQuiesced: PUBLIC ERROR = CODE;
  aMoment: CONDITION;
  csmPriority: Process.Priority ← 5;
  ConvertFaceToChannelTransferStatus: ARRAY TTYPortFace.TransferStatus OF
    TTYPort.TransferStatus =
    [success, parityError, asynchFramingError, dataLost, breakDetected, aborted];
  CharsAvailable: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus]
    RETURNS [number: CARDINAL] =
    BEGIN
    IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN
      ERROR ChannelQuiesced;
    RETURN[channel.charsInBuffer];
    END;

  DeleteChannel: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    DoDeleteChannel[channel];
    AbortStatus[channel];
    JOIN channel.interrupt;
    END;

  Get: PUBLIC PROCEDURE [channel: LONG POINTER TO TTYPortInternal.ChannelStatus]
    RETURNS [data: CHARACTER, status: TTYPort.TransferStatus] =
    BEGIN
    IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN
      ERROR ChannelQuiesced;
    WHILE channel.charsInBuffer = 0 DO
      WaitOnInputBuffer[channel];
      IF channel.deleteInProgress THEN
        RETURN[data ← 0C, status ← abortedByDelete];
      IF channel.deviceStatus.aborted THEN RETURN[data ← 0C, status ← aborted];
      ENDLOOP;
    [data, status] ← GetCharFromBuff[channel];
    END;

  GetStatus: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus]
    RETURNS [stat: TTYPort.DeviceStatus] =
    BEGIN
```



```
    IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN
      ERROR ChannelQuiesced;
    UpdateDeviceStatus[channel];
    RETURN[channel.deviceStatus];
    END;

  InitializeInterrupts: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN channel.interrupt ← FORK InterruptProcess[channel]; END;

  Put: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus, data: CHARACTER]
    RETURNS [status: TTYPort.TransferStatus] =
    BEGIN
    faceTS: TTYPortFace.TransferStatus ← notReady;
    IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN
      ERROR ChannelQuiesced;
    WHILE faceTS = notReady DO
      WHILE ~channel.deviceStatus.readyToPut DO
        UpdateDeviceStatus[channel];
        IF channel.deviceStatus.readyToPut THEN EXIT;
        WaitStatusChange[channel]; -- wait at most 10 secs
        IF channel.deleteInProgress THEN RETURN[status ← abortedByDelete];
        IF channel.deviceStatus.aborted THEN RETURN[status ← aborted];
        ENDLOOP;
      faceTS ← TTYPortFace.PutCommand[channel.lineNumber, data];
      ENDLOOP;
    IF faceTS = breakDetected AND channel.breakBeingProcessed THEN
      BreakDone[channel];
    IF channel.deleteInProgress THEN RETURN[status ← abortedByDelete];
    IF channel.deviceStatus.aborted THEN RETURN[status ← aborted];
    RETURN[status ← success];
    END;

  Quiesce: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN
      ERROR ChannelQuiesced;
    AbortStatus[channel];
    END;

  SetParameter: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus,
    parameter: TTYPort.Parameter] =
    BEGIN
    IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN
      ERROR ChannelQuiesced;
    DoSetParameter[channel, parameter];
    END;

  StatusWait: PUBLIC PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus,
    stat: TTYPort.DeviceStatus] RETURNS [newstat: TTYPort.DeviceStatus] =
    -- Note deletion of the channel subsequent to calling StatusWait causes StatusWait to return n
**ormally.
    BEGIN
    IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN
```

```
    ERROR ChannelQuiesced;
  IncrementStatusWaitCount[channel];
  WHILE ~channel.deleteInProgress AND stat = channel.deviceStatus DO
    WaitStatusChange[channel]; -- max wait of 10 secs

    ENDLOOP;
  DecrementStatusWaitCount[channel];
  RETURN[channel.deviceStatus];
  END;

AbortStatus: PRIVATE ENTRY PROCEDURE [
  channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  channel.deviceStatus.aborted ← TRUE;
  WHILE channel.statusWaitCount > 0 DO
    BROADCAST channel.statusChange;
    WAIT aMoment; -- wait for 50 milliseconds

    ENDLOOP;
  channel.deviceStatus ← [, FALSE, FALSE, FALSE, FALSE, FALSE];
  END;

BreakDone: PRIVATE ENTRY PROCEDURE [
  channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  channel.breakBeingProcessed ← FALSE;
  END;

DecrementStatusWaitCount: PRIVATE ENTRY PROCEDURE [
  channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  channel.statusWaitCount ← channel.statusWaitCount - 1;
  END;

DoDeleteChannel: PRIVATE ENTRY PROCEDURE [
  channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  channel.deleteInProgress ← TRUE;
  BROADCAST channel.wakeUpPtr↑;
  BROADCAST channel.bufferInput;
  channel.parameterRecord.dataSetReady ← FALSE;
  TTYPortFace.SetParameter[channel.lineNumber, [dataSetReady[FALSE]]];
  END;

DoSetParameter: PRIVATE ENTRY PROCEDURE [
  channel: LONG POINTER TO TTYPortInternal.ChannelStatus,
  parameter: TTYPort.Parameter] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  WITH parameter SELECT FROM
    breakDetectedClear =>
      BEGIN
      IF channel.breaksInBuffer = 0 THEN -- all breaks have been seen
        BEGIN
        channel.deviceStatus.breakDetected ← FALSE;
```



```
        BROADCAST channel.statusChange;
        END;
      END;
    characterLength =>
      BEGIN
      channel.parameterRecord.characterLength ← characterLength;
      TTYPortFace.SetParameter[
        channel.lineNumber, [characterLength[characterLength]]];
      END;
    clearToSend =>
      BEGIN
      channel.parameterRecord.clearToSend ← clearToSend;
      TTYPortFace.SetParameter[channel.lineNumber, [clearToSend[clearToSend]]];
      END;
    dataSetReady =>
      BEGIN
      channel.parameterRecord.dataSetReady ← dataSetReady;
      TTYPortFace.SetParameter[
        channel.lineNumber, [dataSetReady[dataSetReady]]];
      END;
    lineSpeed =>
      BEGIN
      channel.parameterRecord.lineSpeed ← lineSpeed;
      TTYPortFace.SetParameter[channel.lineNumber, [lineSpeed[lineSpeed]]];
      END;
    parity =>
      BEGIN
      channel.parameterRecord.parity ← parity;
      TTYPortFace.SetParameter[channel.lineNumber, [parity[parity]]];
      END;
    stopBits =>
      BEGIN
      channel.parameterRecord.stopBits ← stopBits;
      TTYPortFace.SetParameter[channel.lineNumber, [stopBits[stopBits]]];
      END;
    ENDCASE;
  END;

GetCharFromBuff: PRIVATE ENTRY PROCEDURE [
  channel: LONG POINTER TO TTYPortInternal.ChannelStatus]
  RETURNS [data: CHARACTER, status: TTYPort.TransferStatus] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  channel.myDataLost ← FALSE;
  IF channel.buffer[channel.topInUse].stat = breakDetected THEN
    channel.breaksInBuffer ← channel.breaksInBuffer - 1;
  [data, status] ← channel.buffer[channel.topInUse];
  channel.charsInBuffer ← channel.charsInBuffer - 1;
  channel.topInUse ← (channel.topInUse + 1) MOD TTYPortInternal.maxBufferSize;
  END;

IncrementStatusWaitCount: PRIVATE ENTRY PROCEDURE [
  channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
  BEGIN
  ENABLE UNWIND => NULL; -- Required in order to release monitor lock
  channel.statusWaitCount ← channel.statusWaitCount + 1;
  END;

InterruptProcess: PROCEDURE [
```

```
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    Process.SetPriority[csmPriority];
    DO
      WaitForWakeup[channel];
      UpdateDeviceStatus[channel];
      IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN RETURN;
      IF channel.deviceStatus.readyToGet THEN
        BEGIN
        data: CHARACTER;
        faceTS: TTYPortFace.TransferStatus;
        status: TTYPort.TransferStatus;
        [data, faceTS] ← TTYPortFace.GetCommand[channel.lineNumber];
        IF faceTS = notReady THEN ERROR;
        status ← ConvertFaceToChannelTransferStatus[faceTS];
        IF channel.deleteInProgress THEN status ← abortedByDelete;
        IF channel.deviceStatus.aborted THEN status ← aborted;
        IF (data = 0C AND channel.breakBeingProcessed) AND status # breakDetected
          THEN BreakDone[channel]
        ELSE PutCharInBuff[channel, data, status];
        IF channel.deleteInProgress OR channel.deviceStatus.aborted THEN RETURN;
        END;
      ENDLOOP;
    END;

  PutCharInBuff: PRIVATE ENTRY PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus, data: CHARACTER,
    status: TTYPort.TransferStatus] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    IF status # breakDetected THEN channel.breakBeingProcessed ← FALSE
    ELSE IF channel.breakBeingProcessed THEN RETURN;
    -- don't do anything with continuing break
    IF channel.charsInBuffer < TTYPortInternal.maxBufferSize THEN
      BEGIN
      IF status = breakDetected THEN -- new break seen
        BEGIN
        channel.breaksInBuffer ← channel.breaksInBuffer + 1;
        channel.breakBeingProcessed ← TRUE;
        channel.deviceStatus.breakDetected ← TRUE;
        BROADCAST channel.statusChange;
        END;
      channel.buffer[channel.topUnused] ← [data, status];
      channel.charsInBuffer ← channel.charsInBuffer + 1;
      channel.topUnused ←
        (channel.topUnused + 1) MOD TTYPortInternal.maxBufferSize;
      END
    ELSE -- no room in buffer
      IF ~channel.myDataLost THEN -- only do dataLost flagging once
        BEGIN
        channel.myDataLost ← TRUE;
        channel.buffer[
          (channel.topUnused + TTYPortInternal.maxBufferSize - 1) MOD
            TTYPortInternal.maxBufferSize].stat ← dataLost;
        END;
    BROADCAST channel.bufferInput;
    END;

  UpdateDeviceStatus: PRIVATE ENTRY PROCEDURE [
```



```
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    faceDS: TTYPortFace.DeviceStatus ← TTYPortFace.GetStatus[channel.lineNumber];
    channel.deviceStatus.dataTerminalReady ← faceDS.dataTerminalReady;
    channel.deviceStatus.readyToGet ← faceDS.readyToGet;
    channel.deviceStatus.readyToPut ← faceDS.readyToPut;
    channel.deviceStatus.requestToSend ← faceDS.requestToSend;
    NOTIFY channel.statusChange;
    END;

  WaitForWakeup: PRIVATE ENTRY PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    WAIT channel.wakeUpPtr; -- wait for microcode event (naked notify)

    END;

  WaitOnInputBuffer: PRIVATE ENTRY PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    WAIT channel.bufferInput; -- max wait 10 seconds

    END;

  WaitStatusChange: PRIVATE ENTRY PROCEDURE [
    channel: LONG POINTER TO TTYPortInternal.ChannelStatus] =
    BEGIN
    ENABLE UNWIND => NULL; -- Required in order to release monitor lock
    WAIT channel.statusChange; -- max wait 10 seconds

    END;
  -- MAIN PROGRAM

    Process.SetTimeout[@aMoment, Process.MsecToTicks[50]];
    END. -- TTYPortDriverB
```

LOG

Time: July 18, 1980 4:21 PM By: Mary Artibee Action: Use TTYPortFace; changes to Param
***eter, DeviceStatus, TransferStatus.*
Time: July 28, 1980 6:47 PM By: Mary Artibee Action: Split into 2 driver monitors; this is the
***monitor for TTYPort ChannelStatus record.*
Time: July 30, 1980 3:04 AM By: Mary Artibee Action: Changes to StatusWait, DeleteChan
***nel, InitializeInterrupts.*
Time: August 4, 1980 2:07 PM By: Mary Artibee Action: Changed BroadcastDSChange; use
***maxBufferSize in GetCharFromBuff/PutCharInBuff.*
Time: August 5, 1980 5:03 PM By: Mary Artibee Action: Remove timeout on Get/Put, loop on
***notReady TS in Get/Put; added ResidentHeap condition.*
Time: August 28, 1980 8:38 PM By: Mary Artibee Action: Fix lost wakeup bug.
Time: September 3, 1980 12:29 AM By: Mary Artibee Action: Remove ResidentHeap.fir
***st64K, because of ProcessInternal.(de)AllocateNakedCondition.*
Time: September 15, 1980 2:37 PM By: Mary Artibee Action: SetParameter does Broad
***cast on StatusChange.*
Time: October 2, 1980 8:01 AM By: Mary Artibee Action: Handle dataLost, breakDetected.
Time: October 6, 1980 4:37 PM By: Mary Artibee Action: Upped priority of InterruptProcess.

Alto/Mesa Statistics Package 5.0

Statistics as of 13-Oct-80 13:47:25

	chars	lines	code bytes	frame size	ngfi	nlinks	code pages	symbol pages
Checksum	3174	63						6
Dialup	1262	15						4
Keystations	2894	72						6
System	5718	57						6
TemporaryTransaction	506	9						3
TextBlt	2644	46						5
Environment	4562	58						6
Transaction	782	14						3
TTYPortEnvironment	966	13						4
PilotClient	466	8						3
Device	786	11						3
Process	2980	42						5
DiagnosticPilotClient	670	10						3
JLevelIVKeys	2358	45						9
Socket	3740	86						7
Keys	2926	55						10
Zone	3380	35						6
BitBlt	3478	58						6
LevelIIIKeys	2436	41						9
TTYPort	3502	50						7
LevelIVKeys	2228	40						9
Stream	8640	102						10
ByteBlt	1006	12						3
DeviceTypes	1162	17						3
RS232CEnvironment	1714	24						5
Inline	3916	44						7
NetworkStream	3554	74						7
RS232CCorrespondents	2628	31						4
File	5788	65						8
RS232C	5776	69						10
FileTypes	4018	40						4
Space	5300	61						9
RS232CManager	3348	44						7
Runtime	2688	63						7
SubSys	426	20						3
Volume	3650	46						6
TemporaryBooting	3818	38						7
PhysicalVolume	6232	60						9
Scavenger	3822	46						7
Heap	6278	72						9
VolumeExtras	472	9						3
UserTerminal	1542	57						5
FloppyChannel	8084	103						7
PilotSwitches	958	12						3
VMMapLog	4674	60						5
CPSwapDefs	2710	115						8
SpecialSystem	2822	34						4
PSB	2866	98						7
Trap	266	13						3
PrincOpsRuntime	1176	39						4
Frame	1216	34						5
PilotLoadStateFormat	1248	48						4
CountPrivate	1200	44						5
PageMap	4474	48						5
PerfPrivate	2852	104						9
ProcessOperations	2150	65						7
PilotLoadStateOps	1954	53						7
BootFile	5556	77						5
PerfStructures	1016	28						6
PhoneNetwork	1468	18						4
OISCPTypes	3662	85						12
DriverTypes	4774	155						11
PupTypes	4136	133						10
CommUtilDefs	1642	47						5
StatsDefs	6224	166						18
MiniEthernetDefs	1250	43						5

BufferDefs	8412	229						19
StatsOps	492	18						3
SpecialCommunication	1648	37						4
TeledebugProtocol	3820	70						6
PupDefs	5916	146						15
DriverDefs	6452	169						15
OISCPDefs	4148	73						9
TTYPortFace	2250	38						6
DisplayFace	6582	49						5
EthernetOneFace	5148	57						6
RS366Face	1630	20						4
ProcessorFace	7778	69						5
RS232CFace	4892	51						8
PilotDisk	6442	89						10
MouseFace	1178	18						4
KeyboardFace	586	10						3
EthernetFace	4818	53						6
SA4000Face	9272	74						7
SA800Face	10340	128						7
Utilities	3876	46						7
TTYPortInternal	2930	47						5
TemporarySetGMT	778	11						3
StoreDriverStartChain	1416	14						3
Boot	4572	57						7
DOInputOutput	4122	52						5
SpecialSpace	3648	31						5
DriverStartChain	1406	14						3
SpecialFile	1778	19						4
ProcessPriorities	516	10						3
Snapshot	2046	22						4
ResidentHeap	2298	23						4
FilePageLabel	1772	33						12
RuntimeInternal	2832	39						6
ProcessInternal	1618	20						3
PilotMP	3836	51						4
PilotFileTypes	3464	43						4
SpecialTransaction	2024	17						3
SpecialHeap	726	11						3
HeadStartChain	1402	14						3
SoftwareTextBlt	924	19						3
DeviceCleanup	2316	41						4
DiskChannel	8514	100						9
OISTransporter	908	12						3
SpecialVolume	2440	27						5
BootSwap	6916	123						13
MIOCInternalDO	24048	353						29
DebuggerSwap	1394	30						3
TOTAL:	381938	6121	0	0	0	0	0	714

Alto/Mesa Statistics Package 5.0

Statistics as of 13-Oct-80 15:13:29

	chars	lines	code bytes	frame size	ngfi	nlinks	code pages	symbol pages
BcdOperations	4960	150	660	4	1	0	2	12
BootChannel	2814	31						4
BootChannelDisk	4878	131	340	19	1	9	1	12
bootChannelEther	9274	185	670	17	1	7	2	13
BootSwapCross	3704	73	124	10	1	0	1	9
BootSwapGerm	31172	579	1676	12	1	9	4	31
Boss	7436	222	670	40	1	16	2	11
BufferPoolImpl	16882	379	1766	4	1	16	4	23
CachedRegion	19564	164						11
CachedRegionImplA	24756	354	1334	47	1	10	3	19
CachedRegionImplB	42514	871	3470	161	1	17	7	29
CachedRegionInternal	3914	37						5
CachedSpace	4948	60						6
CachedSpaceImpl	4294	133	340	906	1	2	1	8
Checksums	1110	32						4
ChecksumsImpl	4414	111	310	4	1	0	1	10
CommunicationControl	3168	82	124	5	1	18	1	6
CommunicationInternal	1312	23						3
CommunicationPrograms	598	8						3
ControlPrograms	764	10						3
DiagnosticPilotImpl	6768	180	550	5	1	7	2	18
DiagnosticPilotStubsA	3462	44	120	4	1	1	1	7
DiagnosticPilotTest	2098	34	110	4	1	1	1	5
DialupImpl	7410	121	586	9	1	5	2	6
DiskChannelBackend	4216	51						6
DiskChannelImpl	12986	314	1436	12	1	5	3	17
DiskDriverShared	1666	22						4
DiskDriverSharedImpl	8796	139	510	209	1	1	1*	8
DispatcherImpl	6288	215	578	31	1	10	2	14
Echo	476	23						3
EchoServerImpl	5252	140	348	12	1	17	1	11
EthernetDriver	27090	852	2802	123	1	40	6	37
EthernetHeadDO	10862	364	1142	30	1	4	3	14
EthernetOneDriver	30692	916	2844	120	1	41	6	38
EthernetOneHeadDO	11738	380	1168	30	1	4	3	15
FileCache	1522	21						5
FileCacheImpl	21076	336	1712	20	1	3	4	15
FileImpl	43276	1094	7078	34	2	50	14*	53
FileInternal	2210	35						5
FilePageTransfer	1998	29						4
FilerControl	1216	22	38	4	1	5	1	3
FilerException	682	11						3
FilerExceptionImpl	2766	43	144	8	1	1	1	6
FilerPrograms	762	13						3
FilerTransferImpl	10968	244	1214	27	1	14	3	19
FileTask	2244	24						5
FileTaskImpl	8690	132	540	27	1	4	2	12
FloppyChannelImpl	17658	461	1658	68	1	27	4	27
FloppyChannelInternal	2448	38						5
FMPrograms	2278	22						4
Fonts	1614	34						4
FrameImpl	11762	308	1138	5	1	4	3	20
GMTUsingIntervalTimer	3884	69	182	12	1	4	1	7
halfduplex	916	16						4
halfduplexImpl	14308	303	972	40	1	9	2	19
HeapImpl	20808	578	2150	18	2	32	5	27
Hierarchy	4302	45						5
HierarchyImpl	11194	149	554	19	1	9	2	10
Instructions	8892	278	954	4	1	1	2	16
KernelFile	6258	54						7
KernelSpace	1648	16						3
LabelTransfer	3206	50						6
LogicalVolume	12388	131						13
MapLog	772	12						3
MapLogImpl	8514	109	410	10	1	7	1	13
MarkerPage	3466	42						7

code bytes frame size

MarkerPageImpl	9252	134	1378	73	1	7	3	20
MiniEthernetDriver	8680	270	1188	15	1	10	3	15
MIOCCommCmdsD0	33956	553	2520	50	1	27	5*	34
MIOCCommImplD0	36940	587	2600	65	2	25	6	27
MIOCHardwareD0	21300	325	1312	20	1	8	3	22
MiscPrograms	1066	16						3
MStore	6314	52						6
MStoreImpl	12298	293	1036	278	1	5	3	15
NetworkStreamInstance	12904	332	852	26	1	12	2	17
NetworkStreamInternal	860	29						4
NetworkStreamMgr	10552	239	900	9	1	18	2	20
PacketStream	4744	120						11
PacketStreamInstance	51648	1114	3622	132	1	31	8	34
PacketStreamMgr	6028	136	566	147	1	4	2	10
PageFault	768	12						3
PageFaultImpl	4982	119	364	120	1	0	1	11
PageFaultTestDefs	3646	49						6
PageFaultTestHelper	7348	113	574	19	1	3	2	8
PageFaultTestImpl	41628	739	5074	1190	2	8	10	32
PerformancePrograms	524	9						3
PhoneNetworkDriver	35742	697	2086	75	2	28	5	34
PhoneNetworkImpl	6046	120	406	7	1	5	1	8
PhysicalVolumeFormat	9222	89						8
PhysicalVolumeImpl	39640	880	5504	16	2	48	11	45
PilotCommUtil	5160	150	296	4	1	16	1	13
PilotControl	23122	498	1630	106	1	56	4	34
PilotCounter	5916	176	560	19	1	7	2	15
PilotFloppyFormat	2000	25						4
PilotLoaderCore	19664	557	3508	4	2	33	7	34
PilotLoaderOps	2126	62						8
PilotLoaderSupport	14354	428	2202	16	2	27	5	29
PilotLoadState	6378	173	846	12	1	14	2	14
PilotNub	19578	545	1784	92	1	12	4	37
PilotPerfMonitor	10276	324	1278	806	1	1	3	18
PilotUnLoader	9136	277	1000	5	1	25	2*	21
Processes	12476	342	1316	18	1	2	3	21
ProcessorHeadD0	6968	180	368	21	1	0	1	13
Projection	2658	30						4
ProjectionImpl	9160	138	546	4	1	6	2	11
QueueImpl	5058	131	512	4	1	5	1*	12
RDC	3840	102						8
RemotePageTransfer	1262	20						4
ResidentHeapImpl	11828	142	688	80	1	9	2	13
ResidentMemory	3738	32						5
ResidentMemoryImpl	10772	135	534	28	1	9	2	15
Router	3718	79						9
RouterImpl	19438	459	1418	23	1	21	3	22
RoutingTableImpl	21586	559	2022	21	1	20	4*	23
RS232CDriverA	6054	64	364	6	1	9	1	8
RS232CDriverB	13990	173	1310	5	1	15	3	16
RS232CHeadFrontEndD0	11322	160	572	17	1	6	2	13
RS232CInternal	4128	28						7
RS232CManagerImpl	13328	197	1042	9	1	15	3	12
RuntimePrograms	2506	38						5
SA4000HeadD0	21526	304	1044	9	1	4	3	20
SA4000Impl	13512	335	1080	145	1	23	3	21
SA800HeadD0	22538	481	2192	40	1	4	5	22
SA800Impl	19994	279	1530	32	1	23	3*	26
SA800NeckD0	8928	182						9
ScavengeImpl	21728	542	3538	35	1	37	7*	33
SetGMTUsingEthernet	5662	149	424	4	1	13	1	10
Signals	8574	271	624	4	1	1	2	12
SimpleSpace	5580	56						7
SimpleSpaceImpl	21172	260	1550	21	1	14	4	22
SnapshotImpl	10334	247	1368	7	1	25	3	26
SocketImpl	16958	383	1436	14	1	28	3	23
SocketInternal	2746	59						6
SpaceImplA	41120	509	2344	24	2	28	5	31
SpaceImplB	36322	432	2254	5	1	27	5	29
SpaceImplInternal	3330	37						5
SpecialTransactionStub	138	5	18	4	1	0	1	3
StartChainPlug	744	12	18	4	1	0	1	3
StatsHot	978	27	54	447	1	0	1	3
STLeaf	3686	49						6
STLeafImpl	9962	245	1062	49	1	14	3	18
StoragePrograms	10638	96						10

StreamImpl	11706	176	912	15	1	1	2	14
STree	1390	22						5
STreeImpl	19892	360	1406	18	1	8	3	16
SubVolume	4206	54						6
SubVolumeImpl	12640	185	1042	184	1	7	3	16
SwapBuffer	800	12						3
SwapBufferImpl	4498	125	428	10	1	4	1	11
SwapperControl	5592	78	192	18	1	14	1	8
SwapperException	976	12						4
SwapperExceptionImpl	3018	52	134	8	1	1	1	5
SwapperPrograms	2162	26						3
SwapTask	2582	27						4
SwapTaskImpl	3724	55	188	6	1	1	1	6
System6TablesD0	33936	427	2436	4	1	0	5	12
SystemBufferPoolImpl	16160	409	1710	35	1	21	4	22
SystemImpl	5820	161	404	9	1	12	1	13
SystemInternal	2546	28						3
TeleDebug	582	14						3
TeleDebugImpl	9982	225	950	619	1	12	2	20
TextBltImpl	5006	126	356	31	1	0	1	10
TransactionImpl	33664	423	3428	11	1	47	7	27
TransactionInternal	13312	111						10
TransactionLogImpl	8438	122	676	5	1	8	2	14
TransactionState	7558	88						7
TransactionStateImpl	17304	227	1306	36	1	8	3	15
Traps	17232	527	1574	263	1	4	4	27
TTYPortDriverA	5580	89	422	8	1	10	1	9
TTYPortDriverB	15624	249	1552	6	1	7	4	15
ttyportheadd0	18376	286	1064	9	1	4	3	15
UserTerminalHeadD0	13456	230	672	34	1	4	2	17
UserTerminalImpl	6964	200	714	43	1	39	2	17
UtilitiesImpl	1806	25	78	4	1	0	1	4
UtilitySpaceImpl	17674	240	1124	31	1	16	3	24
UtilityVMMControl	3790	41	98	4	1	2	1	7
VM	810	12						3
VMMControl	9196	97	256	7	1	16	1	12
VMMgrStore	1288	15						4
VMMPrograms	1908	23						4
VolAllocMap	3862	41						4
VolAllocMapImpl	13378	174	1240	15	1	6	3	17
VolFileMap	3160	37						5
VolFileMapImpl	24678	361	2692	67	1	9	6	22
VolumeImpl	35840	893	5410	77	2	37	11	48
VolumeImplInterface	4012	37						8
VolumeInternal	1676	23						4
X800TablesD0	22002	275	1374	4	1	0	3	9
XferTrap	926	32						4
ZoneImpl	20954	450	2050	6	1	1	5	15
ZoneInternal	2320	36						4
TOTAL:	1973806	38298	152624	8087	131	1497	366	2502

	chars	lines	code bytes	frame size	ngfi	nlinks	code pages	symbol pages	bytes /line
Pilot Defs:	381938	6121	0	0	0	0	0	714	
Pilot Impls:	1973806	38298	152624	8087	131	1497	366	2502	3.99
SubTotal:	2355744	44419	152624	8087	131	1497	366	3216	3.44
Tests/Othello:	664896	16321	123464	10196	49	1100	262	817	7.56
Flex/OISCPTool:	200090	5355	43466	3062	15	325	91	295	8.12
ComSoft:	732206	19328	79434	3221	72	859	192	1110	4.11
GateStream:	809118	10961	46702	2784	34	483	109	646	4.26
OldWisk:	370692	11348	38810	1011	33	569	95	651	3.42
OldTajo:	176962	5398	20816	675	17	462	47	345	3.86
CoPilot:	632256	19699	84694	3350	71	1738	193	1565	4.30
CPPerfTool:	82274	2539	14688	148	8	205	32	188	5.78
Tajo:	611056	18475	82612	809	65	1313	195	1073	4.47
Total	6635294	153843	687310	33343	495	8551	1582	9906	4.47

Alto Based Development Aids

Packager:	374812	11895	44364	1115	32	470	99	560	
PilotBootToEtherBoot:	5406	138	538	63	1	23	2	12	
RunPilot:	10654	326	1206	47	1	21	3	23	
StartPilot:	146330	4852	20992	1732	15	343	48	307	
TOTAL:	537202	17211	67100	2957	49	857	152	902	

BcdDefs is included by

BcdOperations	BcdOps
FrameImpl	PilotLoaderCore
PilotLoaderOps	PilotLoaderSupport
PilotLoadState	PilotLoadStateOps
PilotUnLoader	

BcdOperations is included by nothing

BcdOps is included by

BcdOperations	FrameImpl
PilotLoaderCore	PilotLoaderOps
PilotLoaderSupport	PilotLoadState
PilotLoadStateOps	PilotUnLoader

BitBlt is included by

DisplayFace	StreamImpl
TextBltImpl	UserTerminal
UserTerminalHeadDO	UserTerminalImpl

Boot is included by

BootChannel	BootChannelDisk
BootChannelEther	BootSwap
BootSwapCross	BootSwapGerm
DebuggerSwap	DiagnosticPilotImpl
FileImpl	FMPrograms
LogicalVolume	MarkerPageImpl
PhysicalVolumeFormat	PhysicalVolumeImpl
PilotControl	PilotNub
ScavengeImpl	SnapshotImpl
SpecialVolume	StoragePrograms
VolumeImpl	

BootChannel is included by

BootChannelDisk	BootChannelEther
BootSwapGerm	

BootChannelDisk is included by nothing

BootChannelEther is included by nothing

BootFile is included by

BootSwap	BootSwapGerm
PilotNub	SnapshotImpl

BootSwap is included by

BootSwapCross	BootSwapGerm
PilotControl	PilotNub
SnapshotImpl	

BootSwapCross is included by nothing

BootSwapGerm is included by nothing

Boss is included by nothing

BufferDefs is included by

BufferPoolImpl	Checksums
ChecksumsImpl	DispatcherImpl
DriverDefs	EthernetDriver
EthernetOneDriver	HalfDuplex
HalfDuplexImpl	MiniEthernetDriver
NetworkStreamInstance	NetworkStreamMgr
OISCPDefs	PacketStream
PacketStreamInstance	PacketStreamMgr
PhoneNetworkDriver	PupDefs
QueueImpl	Router
RouterImpl	RoutingTableImpl
SocketImpl	SocketInternal
SystemBufferPoolImpl	

BufferPoolImpl is included by nothing

ByteBlt is included by

MIOCCommImplDO	NetworkStreamInstance
PacketStreamInstance	RouterImpl
StreamImpl	

CachedRegion is included by

CachedRegionImplA	CachedRegionImplB
CachedRegionInternal	DiagnosticPilotImpl
MapLogImpl	Projection
ProjectionImpl	SimpleSpaceImpl
SpaceImplA	SpaceImplB
SpaceImplInternal	STLeaf
STreeImpl	SwapperControl
SwapperException	SwapperExceptionImpl
SwapTask	SwapTaskImpl
UtilitySpaceImpl	VMMControl

CachedRegionImplA is included by nothing

CachedRegionImplB is included by nothing

CachedRegionInternal is included by

CachedRegionImplA	CachedRegionImplB
ResidentMemoryImpl	

CachedSpace is included by

CachedRegion	CachedRegionImplA
CachedRegionImplB	CachedSpaceImpl
DiagnosticPilotImpl	DiagnosticPilotStubsA
FileImpl	Hierarchy
HierarchyImpl	MapLog
MapLogImpl	MarkerPageImpl
PhysicalVolumeImpl	ProjectionImpl
ScavengeImpl	SimpleSpace
SimpleSpaceImpl	SpaceImplA
SpaceImplB	SpaceImplInternal
STLeaf	STLeafImpl
STree	STreeImpl
TransactionLogImpl	TransactionStateImpl
UtilitySpaceImpl	UtilityVMMControl
VMMControl	VMMPrograms
VolAllocMapImpl	VolFileMapImpl
VolumeImpl	

CachedSpaceImpl is included by nothing

Checksum is included by nothing

Checksums is included by

ChecksumsImpl	RouterImpl
RoutingTableImpl	

ChecksumsImpl is included by nothing

CommunicationControl is included by nothing

CommunicationInternal is included by

ChecksumsImpl	CommunicationControl
EchoServerImpl	NetworkStreamMgr
PacketStreamMgr	PilotCommUtil
RouterImpl	RoutingTableImpl
SocketImpl	

CommunicationPrograms is included by

CommunicationControl	DiagnosticPilotImpl
PilotControl	

CommUtilDefs is included by

Boss	BufferPoolImpl
ChecksumsImpl	EthernetDriver
EthernetOneDriver	PhoneNetworkDriver
PilotCommUtil	RouterImpl
SocketImpl	SystemBufferPoolImpl

ControlPrograms is included by

PilotControl	SystemImpl

CountPrivate is included by

PilotCounter	

CPSwapDefs is included by

PilotCounter	PilotNub
PilotPerfMonitor	

DOInputOutput is included by

EthernetHeadDO	EthernetOneHeadDO
MIOCHardwareDO	ProcessorHeadDO
SA4000HeadDO	SA800HeadDO
UserTerminalHeadDO	

DebuggerSwap is included by

PilotControl	PilotNub

Device is included by

Boot	BootChannelDisk
BootChannelEther	DeviceTypes
DiagnosticPilotImpl	DiskChannel
DiskChannelBackend	DiskChannelImpl
FileImpl	FloppyChannelImpl
FMPrograms	KernelFile
PhysicalVolume	PhysicalVolumeImpl
PilotControl	SA4000Impl
SA800Impl	SnapshotImpl
StoragePrograms	TeledebugImpl
TeledebugProtocol	VolumeImpl

DeviceCleanup is included by

EthernetHeadDO	EthernetOneHeadDO
GMTUsingIntervalTimer	MIOCHardwareDO
PilotNub	SA4000HeadDO
SA800HeadDO	SnapshotImpl
SystemImpl	UserTerminalHeadDO

DeviceTypes is included by

BootChannelDisk	BootChannelEther
FloppyChannelImpl	SA4000Impl
SA800Impl	TeledebugImpl

DiagnosticPilotClient is included by

DiagnosticPilotImpl	DiagnosticPilotTest
PageFaultTestImpl	

DiagnosticPilotImpl is included by nothing

DiagnosticPilotStubsA is included by nothing

DiagnosticPilotTest is included by nothing

Dialup is included by

DialupImpl	PhoneNetworkDriver
RS232CManager	

DialupImpl is included by nothing

DiskChannel is included by

DiskChannelBackend	DiskChannelImpl
DiskDriverShared	DiskDriverSharedImpl
FileImpl	FilerTransferImpl
FileTask	FileTaskImpl
FloppyChannelImpl	FloppyChannelInternal
LabelTransfer	MarkerPage
MarkerPageImpl	PhysicalVolumeImpl
SA4000Impl	SA800Impl
ScavengeImpl	SubVolume
SubVolumeImpl	VolumeImpl

DiskChannelBackend is included by

DiskChannelImpl	DiskDriverShared
DiskDriverSharedImpl	FloppyChannelImpl

SA4000Impl SA800Impl

DiskChannelImpl is included by nothing

DiskDriverShared is included by
DiskDriverSharedImpl FloppyChannelImpl
SA4000Impl

DiskDriverSharedImpl is included by nothing

DispatcherImpl is included by nothing

DisplayFace is included by
UserTerminalHeadDO UserTerminalImpl

DriverDefs is included by
Boss BufferPoolImpl
CommunicationControl DispatcherImpl
EchoServerImpl EthernetDriver
EthernetOneDriver HalfDuplexImpl
NetworkStreamMgr PacketStreamInstance
PacketStreamMgr PhoneNetworkDriver
PilotCommUtil QueueImpl
Router RouterImpl
RoutingTableImpl SocketImpl
SystemBufferPoolImpl

DriverStartChain is included by
FloppyChannelImpl PilotControl
SA800Impl StartChainPlug
UserTerminalImpl

DriverTypes is included by
Boss BufferDefs
BufferPoolImpl DriverDefs
EthernetDriver EthernetOneDriver
HalfDuplexImpl MiniEthernetDriver
PhoneNetworkDriver QueueImpl
SpecialCommunication SystemBufferPoolImpl

Echo is included by
CommunicationControl EchoServerImpl

EchoServerImpl is included by nothing

Environment is included by
BitBlt BootChannel
BootChannelDisk BootChannelEther
BootFile BootSwap
BootSwapGerm ByteBlt
CachedRegion CachedRegionImplA
CachedRegionImplB CachedRegionInternal
CachedSpace CachedSpaceImpl
DiagnosticPilotClient DiagnosticPilotImpl
DiagnosticPilotTest DiskChannel
DiskChannelImpl DisplayFace
EchoServerImpl EthernetDriver
EthernetFace EthernetHeadDO
EthernetOneDriver EthernetOneFace
EthernetOneHeadDO FileCacheImpl
FileImpl FilePageLabel
FilePageTransfer FilerExceptionImpl
FilerTransferImpl FileTask
FileTaskImpl FloppyChannelImpl
FrameImpl HalfDuplexImpl
Heap HeapImpl
Hierarchy HierarchyImpl
Inline Instructions
LabelTransfer MapLogImpl
MarkerPage MarkerPageImpl
MiniEthernetDriver MIOCCommCmdsDO
MIOCCommImplDO MIOCHardwareDO
MIOCInternalDO MStore
MStoreImpl NetworkStreamInstance
NetworkStreamMgr PacketStream
PacketStreamInstance PageFault
PageFaultImpl PageFaultTestDefs
PageFaultTestHelper PageFaultTestImpl
PageMap PerfPrivate
PerfStructures PhoneNetworkDriver
PhoneNetworkImpl PhysicalVolumeFormat
PhysicalVolumeImpl PilotCommUtil
PilotControl PilotCounter
Pilotdisk PilotFloppyFormat
PilotLoaderSupport PilotLoadState
PilotLoadStateFormat PilotNub
PilotPerfMonitor PilotUnLoader
Processes ProcessorFace
ProcessorHeadDO Projection
ProjectionImpl RDC
RemotePageTransfer ResidentHeap
ResidentHeapImpl ResidentMemory
ResidentMemoryImpl RouterImpl
RS232C RS232CDriverB
RS232CEnvironment RS232CFace
RS232CHeadFrontEndDO RS232CInternal
RuntimeInternal RuntimePrograms
SA4000Face SA4000HeadDO
SA4000Impl SA800Impl
ScavengeImpl Scavenger
SetGMTUsingEthernet Signals
SimpleSpace SimpleSpaceImpl
Snapshot SnapshotImpl
Socket SocketImpl
Space SpaceImplA
SpaceImplB SpaceImplInternal
SpecialSpace STLeaf
STLeafImpl StoragePrograms
Stream StreamImpl
STree STreeImpl
SubVolume SubVolumeImpl
SwapBuffer SwapBufferImpl
SwapperControl SwapperException
SwapperExceptionImpl SwapTask
SwapTaskImpl System6TablesDO
SystemImpl Teledebug
TeledebugImpl TextBltImpl
TransactionImpl TransactionInternal
TransactionLogImpl TransactionStateImpl
Traps UserTerminalHeadDO
UserTerminalImpl Utilities
UtilitySpaceImpl UtilityVMMControl
VM VMMControl
VMMgrStore VMMPrograms
VolAllocMapImpl VolFileMapImpl
VolumeImpl X800TablesDO
Zone ZoneImpl

EthernetDriver is included by nothing

EthernetFace is included by
EthernetDriver EthernetHeadDO

EthernetHeadDO is included by nothing

EthernetOneDriver is included by nothing

EthernetOneFace is included by
EthernetOneDriver EthernetOneHeadDO
MiniEthernetDriver SetGMTUsingEthernet

EthernetOneHeadDO is included by nothing

File is included by
Boot BootChannelDisk
CachedRegionImplB DiagnosticPilotImpl
FileCache FileCacheImpl
FileImpl FileInternal
FilePageLabel FilePageTransfer
FilerExceptionImpl FilerTransferImpl
FileTask FileTaskImpl
FileTypes KernelFile
LabelTransfer LogicalVolume
MapLogImpl MarkerPage
MarkerPageImpl PhysicalVolumeImpl
PilotControl PilotDisk
PilotFileTypes PilotLoaderSupport
RDC RemotePageTransfer
Runtime SA4000HeadDO
ScavengeImpl Scavenger
SimpleSpaceImpl Snapshot
SnapshotImpl Space
SpaceImplA SpaceImplB
SpecialFile STLeafImpl
SubSys SubVolume
SubVolumeImpl TemporaryBooting
TransactionImpl TransactionInternal
TransactionLogImpl TransactionState
UtilitySpaceImpl VMMControl
VolAllocMapImpl VolFileMap
VolFileMapImpl Volume
VolumeImpl VolumeImplInterface

FileCache is included by
FileCacheImpl FileImpl
FilerTransferImpl FileTaskImpl
VolumeImpl

FileCacheImpl is included by nothing

FileImpl is included by nothing

FileInternal is included by
CachedRegionImplB FileCache
FileCacheImpl FileImpl
FilePageTransfer FilerExceptionImpl
FilerTransferImpl FileTask
FileTaskImpl LabelTransfer
MarkerPageImpl PhysicalVolumeImpl
RemotePageTransfer ScavengeImpl
SubVolume SubVolumeImpl
TransactionLogImpl VolAllocMap
VolAllocMapImpl VolFileMap
VolFileMapImpl VolumeImpl
VolumeImplInterface

FilePageLabel is included by
FilerTransferImpl → FileTask
FileTaskImpl → SubVolumeImpl

FilePageTransfer is included by
CachedRegionImplB FileImpl
FilerException FilerExceptionImpl
FilerTransferImpl FileTaskImpl
RemotePageTransfer

FilerControl is included by nothing

FilerException is included by
FileImpl FilerExceptionImpl
FilerTransferImpl

FilerExceptionImpl is included by nothing

FilerPrograms is included by
FileCacheImpl FilerControl
FilerExceptionImpl FilerTransferImpl
FileTaskImpl SubVolumeImpl

FilerTransferImpl is included by nothing

FileTask is included by

```
  FilerTransferImpl     FileTaskImpl
  SubVolumeImpl

FileTaskImpl is included by nothing

FileTypes is included by
  SnapshotImpl

FloppyChannel is included by
  FloppyChannelImpl     FloppyChannelInternal
  PilotFloppyFormat     SA800Impl

FloppyChannelImpl is included by nothing

FloppyChannelInternal is included by
  FloppyChannelImpl     SA800Impl

FMPrograms is included by
  FileImpl              MarkerPageImpl
  PhysicalVolumeImpl    ScavengeImpl
  VolAllocMapImpl       VolFileMapImpl
  VolumeImpl

Fonts is included by
  TextBltImpl

Frame is included by
  BootSwap              BootSwapCross
  BootSwapGerm          CachedRegionImplA
  CachedSpaceImpl       FilerExceptionImpl
  FrameImpl             Instructions
  MStoreImpl            PageFaultImpl
  PageFaultTestImpl     PilotControl
  PilotCounter          PilotLoaderCore
  PilotNub              PilotPerfMonitor
  Processes             ProcessorHeadDO
  ResidentMemoryImpl    Signals
  SnapshotImpl          SpaceImplB
  SwapperExceptionImpl  SwapTaskImpl
  Traps                 UtilitySpaceImpl

FrameImpl is included by nothing

GMTUsingIntervalTimer is included by nothing

HalfDuplex is included by
  HalfDuplexImpl        PhoneNetworkDriver

HalfDuplexImpl is included by nothing

HeadStartChain is included by
  BootSwapGerm          EthernetHeadDO
  EthernetOneHeadDO     GMTUsingIntervalTimer
  PilotControl          SA4000HeadDO
  SA800HeadDO           StartChainPlug
  UserTerminalHeadDO

Heap is included by
  EchoServerImpl        EthernetDriver
  EthernetOneDriver     HeapImpl
  MIOCCommCmdsDO        MIOCCommImplDO
  PhoneNetworkImpl      PilotLoaderSupport
  RoutingTableImpl      RS232CDriverA
  RS232CManagerImpl     SocketImpl
  TTYPortDriverA

HeapImpl is included by nothing

Hierarchy is included by
  HierarchyImpl         SpaceImplA
  SpaceImplB            VMMControl

HierarchyImpl is included by nothing

Inline is included by
  BootChannelDisk       BootSwap
  BootSwapCross         BootSwapGerm
  BufferPoolImpl        CachedRegionImplA
  CachedRegionImplB     ChecksumsImpl
  DiagnosticPilotTest   EthernetDriver
  EthernetHeadDO        EthernetOneDriver
  EthernetOneHeadDO     File
  FileImpl              FilePageLabel
  FilerTransferImpl     FloppyChannelImpl
  FrameImpl             HeapImpl
  Instructions          MapLogImpl
  MarkerPageImpl        MiniEthernetDriver
  MIOCCommCmdsDO        MIOCCommImplDO
  MIOCHardwareDO        MStoreImpl
  PacketStreamInstance  PageFaultTestImpl
  PerfPrivate           PerfStructures
  PhysicalVolumeFormat  PhysicalVolumeImpl
  PilotCommUtil         PilotControl
  PilotDisk             PilotLoaderCore
  PilotLoaderSupport    PilotLoadState
  PilotNub              PilotPerfMonitor
  PilotUnLoader         Processes
  ProcessorHeadDO       ResidentHeapImpl
  ResidentMemoryImpl    RouterImpl
  SA4000HeadDO          SA4000Impl
  SA800HeadDO           SA800Impl
  ScavengeImpl          SetGMTUsingEthernet
  SimpleSpace           SimpleSpaceImpl
  SnapshotImpl          SocketImpl
  SpaceImplA            SpaceImplB
  STLeafImpl            StoragePrograms
  StreamImpl            STreeImpl
  SubVolumeImpl         SwapBufferImpl
  SystemBufferPoolImpl  SystemImpl
  TeledebugImpl         TransactionImpl
  TransactionInternal   TransactionLogImpl
  TransactionStateImpl  Traps
  UserTerminalHeadDO    Utilities
  UtilitiesImpl         UtilitySpaceImpl
  VolAllocMapImpl       VolFileMapImpl
  VolumeImpl            ZoneImpl

Instructions is included by nothing

JLevelIVKeys is included by nothing

KernelFile is included by
  DiagnosticPilotImpl   FileImpl
  MapLogImpl            PilotControl
  ScavengeImpl          SnapshotImpl
  SpaceImplB            STLeafImpl
  TransactionImpl       VolFileMapImpl
  VolumeImpl

KernelSpace is included by
  SpaceImplB            TransactionImpl

KeyboardFace is included by
  PilotNub              UserTerminalHeadDO
  UserTerminalImpl

Keys is included by
  PilotNub

keystations is included by
  JLevelIVKeys          KeyboardFace
  Keys                  LevelIIIKeys
  LevelIVKeys           MouseFace
  PilotNub              UserTerminalHeadDO
  UserTerminalImpl

LabelTransfer is included by
  FileImpl              FilerTransferImpl
  MarkerPageImpl        PhysicalVolumeImpl
  ScavengeImpl          VolAllocMapImpl
  VolumeImpl

LevelIIIKeys is included by nothing

LevelIVKeys is included by nothing

LogicalVolume is included by
  FileImpl              MarkerPage
  MarkerPageImpl        PhysicalVolumeImpl
  ScavengeImpl          VolAllocMap
  VolAllocMapImpl       VolFileMap
  VolFileMapImpl        VolumeImpl
  VolumeImplInterface

LongBases is included by
  BcdDefs               BcdOperations
  BcdOps                PilotLoaderCore
  PilotLoaderOps        PilotLoaderSupport
  PilotUnLoader         Table

MapLog is included by
  MapLogImpl            SpaceImplB
  VMMControl

MapLogImpl is included by nothing

MarkerPage is included by
  MarkerPageImpl        PhysicalVolumeImpl
  ScavengeImpl          VolumeImpl

MarkerPageImpl is included by nothing

MiniEthernetDefs is included by
  BootChannelEther      MiniEthernetDriver
  TeledebugImpl

MiniEthernetDriver is included by nothing

MIOCCommCmdsDO is included by nothing

MIOCCommImplDO is included by nothing

MIOCHardwareDO is included by nothing

MIOCInternalDO is included by
  MIOCCommCmdsDO        MIOCCommImplDO
  MIOCHardwareDO        RS232CHeadFrontEndDO
  System6TablesDO       TTYPortHeadDO
  X800TablesDO

MiscAlpha is included by
  Checksum              Checksums
  PageMap               ProcessorFace
  ProcessorHeadDO       System
  TextBlt               TransactionInternal

MiscPrograms is included by
  DiagnosticPilotImpl   HeapImpl
  PilotControl          ResidentHeapImpl
  StreamImpl            UtilitiesImpl
  ZoneImpl

Mopcodes is included by
  BitBlt                Boot
  BootSwap              BootSwapCross
  BootSwapGerm          Checksum
  Checksums             ChecksumsImpl
  CommUtilDefs          DOInputOutput
  DiagnosticPilotStubsA EthernetHeadDO
  EthernetOneHeadDO     Frame
  FrameImpl             Inline
  Instructions          MIOCHardwareDO
```

```
  PageFaultTestImpl     PageMap
  PilotCounter          PilotLoaderCore
  PilotLoaderSupport    PilotNub
  PilotPerfMonitor      PilotUnLoader
  PrincOpsRuntime       ProcessInternal
  ProcessOperations     ProcessorFace
  ProcessorHeadDO       Runtime
  RuntimeInternal       SA800NeckDO
  Signals               System
  TextBlt               TransactionInternal
  Trap                  Traps
  Utilities             XferTrap

MouseFace is included by
  UserTerminalHeadDO    UserTerminalImpl

MStore is included by
  CachedRegionImplA     CachedRegionImplB
  DiagnosticPilotImpl   FileImpl
  MStoreImpl            ResidentMemoryImpl
  SimpleSpaceImpl       SwapperControl

MStoreImpl is included by nothing

NetworkStream is included by
  NetworkStreamMgr      PacketStream
  PacketStreamInstance  PacketStreamMgr

NetworkStreamInstance is included by
  NetworkStreamMgr

NetworkStreamInternal is included by
  NetworkStreamInstance NetworkStreamMgr

NetworkStreamMgr is included by nothing

OISCPDefs is included by
  Boss                  BufferPoolImpl
  CommunicationControl  EchoServerImpl
  NetworkStreamInstance NetworkStreamMgr
  PacketStream          PacketStreamInstance
  PacketStreamMgr       QueueImpl
  Router                RouterImpl
  RoutingTableImpl      SocketImpl
  SocketInternal        SystemBufferPoolImpl

OISCPTypes is included by
  BufferDefs            ChecksumsImpl
  EchoServerImpl        EthernetDriver
  EthernetOneDriver     NetworkStreamMgr
  OISCPDefs             PacketStream
  PacketStreamInstance  PacketStreamMgr
  PhoneNetworkDriver    Router
  RouterImpl            RoutingTableImpl
  SocketImpl            SocketInternal

OISTransporter is included by
  PhoneNetworkDriver    RS232CManagerImpl

PacketStream is included by
  NetworkStreamInstance NetworkStreamInternal
  NetworkStreamMgr      PacketStreamInstance
  PacketStreamMgr

PacketStreamInstance is included by
  PacketStreamMgr

PacketStreamMgr is included by nothing

PageFault is included by
  CachedRegionImplB     PageFaultImpl
  SwapperControl

PageFaultImpl is included by nothing

PageFaultTestDefs is included by
  PageFaultTestHelper   PageFaultTestImpl

PageFaultTestHelper is included by nothing

PageFaultTestImpl is included by nothing

PageMap is included by
  BootFile              BootSwapGerm
  CachedRegionImplA     CachedRegionImplB
  MStore                MStoreImpl
  SimpleSpaceImpl       SwapBufferImpl
  SwapTask              SwapTaskImpl
  TeledebugImpl

PerformancePrograms is included by
  DiagnosticPilotImpl   PilotControl
  PilotCounter          PilotPerfMonitor
  UtilityVMMControl

PerfPrivate is included by
  PilotPerfMonitor

PerfStructures is included by
  CountPrivate          PerfPrivate
  PilotCounter          PilotPerfMonitor

PhoneNetwork is included by
  PhoneNetworkDriver    PhoneNetworkImpl

PhoneNetworkDriver is included by nothing

PhoneNetworkImpl is included by nothing

PhysicalVolume is included by
  DiskChannelImpl       FloppyChannel
  FloppyChannelImpl     MarkerPage
  MarkerPageImpl        PhysicalVolumeImpl
  SA800Impl             ScavengeImpl
  SnapshotImpl          SpecialVolume
  VolAllocMapImpl       VolumeImpl
  VolumeImplInterface

PhysicalVolumeFormat is included by
  BootSwapGerm          FileImpl
  MarkerPage            MarkerPageImpl
  PhysicalVolumeImpl    ScavengeImpl
  SubVolume             SubVolumeImpl
  VolumeImpl            VolumeImplInterface

PhysicalVolumeImpl is included by nothing

PilotClient is included by
  DiagnosticPilotImpl   PilotControl

PilotCommUtil is included by nothing

PilotControl is included by nothing

PilotCounter is included by nothing

PilotDisk is included by
  BootChannelDisk       DiskChannel
  DiskChannelBackend    DiskChannelImpl
  FileImpl              FilePageLabel
  FilerTransferImpl     FileTask
  FileTaskImpl          FloppyChannelImpl
  LabelTransfer         PhysicalVolumeImpl
  RDC                   SA4000Face
  SA4000HeadDO          SA4000Impl
  SA800Face             SA800HeadDO
  SA800Impl             ScavengeImpl
  SubVolumeImpl         TeledebugImpl
  VolumeImpl

PilotFileTypes is included by
  FileImpl              LogicalVolume
  MarkerPageImpl        PhysicalVolumeImpl
  SA4000HeadDO          ScavengeImpl
  STLeafImpl            TransactionImpl
  TransactionLogImpl    VolAllocMapImpl
  VolFileMapImpl        VolumeImpl

PilotFloppyFormat is included by
  FloppyChannelImpl     SA800Impl

PilotLoaderCore is included by nothing

PilotLoaderOps is included by
  PilotLoaderCore       PilotLoaderSupport
  PilotLoadState        PilotUnLoader

PilotLoaderSupport is included by nothing

PilotLoadState is included by nothing

PilotLoadStateFormat is included by
  FrameImpl             PilotLoaderCore
  PilotLoadState        PilotLoadStateOps
  PilotUnLoader

PilotLoadStateOps is included by
  PilotLoaderCore       PilotLoaderOps
  PilotLoaderSupport    PilotLoadState
  PilotUnLoader

PilotMP is included by
  BootChannelDisk       BootChannelEther
  BootSwapGerm          GMTUsingIntervalTimer
  PhysicalVolumeImpl    PilotControl
  PilotNub              SnapshotImpl
  TeledebugImpl         TransactionImpl
  Traps

PilotNub is included by nothing

PilotPerfMonitor is included by nothing

PilotSwitches is included by
  CachedSpaceImpl       EthernetDriver
  EthernetOneDriver     FileImpl
  HeapImpl              MapLogImpl
  MStoreImpl            PhysicalVolumeImpl
  PilotControl          PilotNub
  SimpleSpaceImpl       STLeafImpl
  TransactionImpl       UserTerminalImpl
  VolumeImpl

PilotUnLoader is included by nothing

PrincOps is included by
  BcdDefs               BootSwap
  BootSwapCross         BootSwapGerm
  BufferPoolImpl        CachedRegionImplA
  CachedRegionImplB     CommUtilDefs
  CountPrivate          CPSwapDefs
  DiagnosticPilotImpl   DiagnosticPilotStubsA
  Frame                 FrameImpl
  Instructions          MStoreImpl
  PageFaultImpl         PageFaultTestImpl
  PerfPrivate           PerfStructures
  PilotControl          PilotCounter
  PilotLoaderCore       PilotLoaderOps
  PilotLoaderSupport    PilotLoadState
  PilotLoadStateFormat  PilotLoadStateOps
  PilotNub              PilotPerfMonitor
  PilotUnLoader         PrincOpsRuntime
  Processes             ProcessOperations
  ProcessorHeadDO       PSB
```

ResidentMemoryImpl RuntimeInternal
RuntimePrograms Signals
SnapshotImpl SpaceImplA
SpaceImplB StreamImpl
SystemBufferPoolImpl Trap
Traps Utilities
UtilitySpaceImpl XferTrap

PrincOpsRuntime is included by
BootSwapGerm DiagnosticPilotStubsA
FrameImpl PageFaultTestImpl
PilotLoaderCore PilotLoaderSupport
PilotUnLoader SpaceImplA
Traps

Process is included by
BufferPoolImpl CachedRegionImplA
CachedRegionImplB DialupImpl
DiskChannelImpl DispatcherImpl
EthernetDriver EthernetOneDriver
FileCacheImpl FileImpl
FileTaskImpl FloppyChannelImpl
HalfDuplexImpl MIOCCommCmdsDO
MIOCCommImplDO MStoreImpl
PacketStreamInstance PageFaultTestHelper
PageFaultTestImpl PhoneNetworkDriver
PilotCommUtil PilotControl
PilotNub Processes
ProcessPriorities ResidentMemoryImpl
RoutingTableImpl RS232CDriverA
RS232CDriverB RS232CHeadFrontEndDO
RS232CManagerImpl SA4000Impl
SA800Impl SocketImpl
SpaceImplA SubVolumeImpl
SwapperControl SystemBufferPoolImpl
SystemImpl TTYPortDriverA
TTYPortDriverB UserTerminalImpl
UtilitySpaceImpl VolumeImpl
ZoneImpl

Processes is included by nothing

ProcessInternal is included by
CachedRegionImplA EthernetDriver
EthernetOneDriver FloppyChannelImpl
GMTUsingIntervalTimer Instructions
MIOCCommImplDO MStoreImpl
PageFaultImpl PageFaultTestImpl
PilotCounter PilotNub
PilotPerfMonitor Processes
SA4000Impl SnapshotImpl
Traps TTYPortDriverA
UserTerminalImpl

ProcessOperations is included by
BootSwapGerm PageFaultImpl
PerfStructures PilotControl
PilotCounter PilotNub
PilotPerfMonitor Processes
SnapshotImpl

ProcessorFace is included by
BootChannelEther BootSwapGerm
DiagnosticPilotImpl DiagnosticPilotTest
GMTUsingIntervalTimer MStoreImpl
PageFaultTestImpl PhysicalVolumeImpl
PilotControl PilotCounter
PilotNub PilotPerfMonitor
Processes ProcessorHeadDO
SetGMTUsingEthernet SnapshotImpl
SystemImpl TeledebugImpl
TransactionImpl Traps
UserTerminalHeadDO

ProcessorHeadDO is included by nothing

ProcessPriorities is included by
SA4000Impl SA800Impl

Projection is included by
ProjectionImpl SpaceImplA
SpaceImplB VMMControl

ProjectionImpl is included by nothing

PSB is included by
BootSwapGerm CountPrivate
PageFaultImpl PerfPrivate
PerfStructures PilotCounter
PilotNub PilotPerfMonitor
Processes ProcessOperations
SnapshotImpl

PupDefs is included by
Boss BufferPoolImpl
PilotCommUtil QueueImpl
SystemBufferPoolImpl

PupTypes is included by
BootChannelEther BufferDefs
BufferPoolImpl DriverDefs
EthernetDriver EthernetOneDriver
MiniEthernetDefs MiniEthernetDriver
PhoneNetworkDriver PupDefs
SocketImpl SystemBufferPoolImpl
TeledebugImpl TeledebugProtocol

QueueImpl is included by nothing

RDC is included by
SA4000HeadDO

RemotePageTransfer is included by nothing

ResidentHeap is included by
DiskChannelImpl FileCacheImpl
FloppyChannelImpl MIOCCommCmdsDO
MIOCCommImplDO PilotCommUtil
ResidentHeapImpl RS232CHeadFrontEndDO
SA4000Impl SpaceImplB
STLeafImpl STreeImpl
SwapBufferImpl TransactionStateImpl
UserTerminalImpl

ResidentHeapImpl is included by nothing

ResidentMemory is included by
BootChannelEther BootSwapGerm
CachedRegionImplA DiagnosticPilotImpl
FileImpl FilerTransferImpl
FileTaskImpl MapLogImpl
MarkerPageImpl PageFaultTestImpl
PhysicalVolumeImpl ResidentHeapImpl
ResidentMemoryImpl ScavengeImpl
SimpleSpace SimpleSpaceImpl
STLeafImpl SwapBufferImpl
TransactionLogImpl TransactionStateImpl
Traps UtilitySpaceImpl
VolAllocMapImpl VolFileMapImpl
VolumeImpl

ResidentMemoryImpl is included by nothing

Router is included by
CommunicationControl EchoServerImpl
NetworkStreamMgr PacketStreamInstance
PacketStreamMgr RouterImpl
RoutingTableImpl SocketImpl

RouterImpl is included by nothing

RoutingTableImpl is included by nothing

RS232C is included by
HalfDuplex HalfDuplexImpl
OISTransporter PhoneNetworkDriver
RS232CDriverA RS232CDriverB
RS232CInternal RS232CManager
RS232CManagerImpl

RS232Ccorrespondents is included by
MIOCCommCmdsDO RS232CHeadFrontEndDO
RS232CInternal

RS232CDriverA is included by nothing

RS232CDriverB is included by nothing

RS232CEnvironment is included by
HalfDuplex HalfDuplexImpl
MIOCCommCmdsDO MIOCCommImplDO
MIOCInternalDO PhoneNetworkDriver
RS232C RS232Ccorrespondents
RS232CDriverA RS232CDriverB
RS232CFace RS232CHeadFrontEndDO
RS232CInternal RS232CManager
RS232CManagerImpl

RS232CFace is included by
MIOCCommCmdsDO MIOCCommImplDO
MIOCInternalDO RS232CDriverA
RS232CDriverB RS232CHeadFrontEndDO
RS232CInternal RS232CManagerImpl

RS232CHeadFrontEndDO is included by nothing

RS232CInternal is included by
RS232CDriverA RS232CDriverB
RS232CManagerImpl

RS232CManager is included by
OISTransporter PhoneNetwork
PhoneNetworkDriver PhoneNetworkImpl
RS232CManagerImpl

RS232CManagerImpl is included by nothing

RS366Face is included by
DialupImpl MIOCHardwareDO

Runtime is included by
DiagnosticPilotImpl DiagnosticPilotStubsA
EthernetDriver EthernetOneDriver
FileImpl FrameImpl
HeapImpl MIOCCommCmdsDO
MIOCCommImplDO MIOCHardwareDO
NetworkStreamInstance PacketStreamInstance
PacketStreamMgr PageFaultTestImpl
PhysicalVolumeImpl PilotCommUtil
PilotControl PilotLoaderSupport
PilotNub PilotUnLoader
SpaceImplA SpaceImplB
STLeafImpl SwapBufferImpl
SwapperControl TransactionImpl
Traps UserTerminalImpl
UtilitySpaceImpl

RuntimeInternal is included by
CachedRegionImplA CachedSpaceImpl
DiskChannelImpl FilerExceptionImpl
FileTaskImpl FrameImpl
Instructions MapLogImpl
MIOCHardwareDO MStoreImpl

PageFaultImpl PageFaultTestImpl
PilotLoaderCore PilotLoaderSupport
PilotLoadState PilotNub
PilotUnLoader ResidentHeapImpl
ResidentMemoryImpl SA4000Impl
SA800Impl Signals
SpaceImplA SubVolumeImpl
SwapBufferImpl SwapperExceptionImpl
SwapTaskImpl SystemImpl
Traps UserTerminalHeadDO

RuntimePrograms is included by
DiagnosticPilotImpl FrameImpl
Instructions PilotControl
PilotLoaderSupport PilotLoadState
PilotNub Processes
Signals SnapshotImpl
Traps UtilityVMMControl

SA4000Face is included by
BootChannelDisk RDC
SA4000HeadDO SA4000Impl
TeledebugImpl

SA4000HeadDO is included by nothing

SA4000Impl is included by nothing

SA800Face is included by
FloppyChannelImpl FloppyChannelInternal
SA800HeadDO SA800Impl
SA800NeckDO

SA800HeadDO is included by nothing

SA800Impl is included by nothing

SA800NeckDO is included by
SA800HeadDO

ScavengeImpl is included by nothing

Scavenger is included by
ScavengeImpl VolumeImpl

SDDefs is included by
BootSwap BootSwapGerm
DiagnosticPilotImpl DiagnosticPilotStubsA
Frame FrameImpl
Instructions PageFaultImpl
PageFaultTestImpl PilotControl
PilotCounter PilotLoadState
PilotNub PilotPerfMonitor
Processes ProcessorHeadDO
Runtime RuntimeInternal
Signals SnapshotImpl
SpaceImplB Traps
UtilitySpaceImpl

SetGNTUsingEthernet is included by nothing

Signals is included by nothing

SimpleSpace is included by
CachedRegionImplA DiagnosticPilotImpl
FileImpl MapLogImpl
MarkerPageImpl PhysicalVolumeImpl
ResidentMemoryImpl ScavengeImpl
SimpleSpaceImpl STLeafImpl
SwapBufferImpl TransactionImpl
TransactionInternal TransactionLogImpl
TransactionStateImpl UtilitySpaceImpl
VMMControl VolAllocMapImpl
VolFileMapImpl VolumeImpl

SimpleSpaceImpl is included by nothing

Snapshot is included by
SnapshotImpl

SnapshotImpl is included by nothing

Socket is included by
EchoServerImpl NetworkStreamMgr
PacketStreamInstance RouterImpl
SocketImpl SocketInternal

SocketImpl is included by nothing

SocketInternal is included by
CommunicationControl NetworkStreamMgr
PacketStreamInstance Router
RouterImpl SocketImpl

SoftwareTextBlt is included by
TextBltImpl

Space is included by
CachedRegionImplA CachedRegionImplB
CachedSpace DiagnosticPilotImpl
EchoServerImpl FileImpl
FloppyChannelImpl Heap
HeapImpl HierarchyImpl
KernelSpace LogicalVolume
MapLogImpl MarkerPageImpl
MIOCCommCmdsDO MIOCCommImplDO
PhoneNetworkImpl PhysicalVolumeImpl
PilotCommUtil PilotControl
PilotCounter PilotLoaderSupport
PilotLoadState PilotUnLoader
ResidentMemoryImpl SA800Impl
ScavengeImpl Scavenger
SimpleSpace SimpleSpaceImpl
SnapshotImpl SpaceImplA
SpaceImplB SpaceImplInternal
SpecialSpace STLeafImpl
StoragePrograms STreeImpl
SwapBufferImpl TransactionImpl
TransactionInternal TransactionLogImpl
TransactionState TransactionStateImpl
UserTerminalImpl UtilitySpaceImpl
VMMControl VMMgrStore
VolAllocMapImpl VolFileMapImpl
VolumeImpl

SpaceImplA is included by nothing

SpaceImplB is included by nothing

SpaceImplInternal is included by
SpaceImplA SpaceImplB

SpecialCommunication is included by
Boss Router
RouterImpl RoutingTableImpl

SpecialFile is included by
FileImpl KernelFile
SnapshotImpl

Specialheap is included by
HeapImpl MIOCCommCmdsDO
MIOCCommImplDO

SpecialSpace is included by
CachedRegionImplA DiagnosticPilotImpl
FloppyChannelImpl HeapImpl
MIOCHardwareDO MStoreImpl
PilotCommUtil PilotCounter
PilotLoaderSupport SA800Impl
SpaceImplA SpaceImplB
SwapBufferImpl UserTerminalImpl
UtilitySpaceImpl

SpecialSystem is included by
Boss BufferDefs
DriverDefs DriverTypes
EchoServerImpl EthernetDriver
EthernetFace EthernetHeadDO
EthernetOneDriver HalfDuplex
HalfDuplexImpl NetworkStreamMgr
OISCPDefs OISCPTypes
PacketStream PacketStreamInstance
PacketStreamMgr PhoneNetwork
PhoneNetworkDriver PhoneNetworkImpl
PilotControl Router
RouterImpl RoutingTableImpl
SA4000HeadDO SocketImpl
SocketInternal SpecialCommunication
SystemImpl SystemInternal

SpecialTransaction is included by
SpecialTransactionStub TransactionImpl
TransactionInternal TransactionStateImpl

SpecialTransactionStub is included by nothing

SpecialVolume is included by
FileImpl PhysicalVolumeImpl
SnapshotImpl VolumeImpl

StartChainPlug is included by nothing

StartList is included by
PilotControl SimpleSpaceImpl
StoragePrograms

StatsDefs is included by
Boss BufferPoolImpl
DispatcherImpl DriverDefs
EchoServerImpl EthernetDriver
EthernetOneDriver PacketStreamInstance
QueueImpl RouterImpl
RoutingTableImpl StatsHot
StatsOps SystemBufferPoolImpl

StatsHot is included by nothing

StatsOps is included by
CommunicationControl StatsHot

STLeaf is included by
HierarchyImpl ProjectionImpl
STLeafImpl STree
STreeImpl VMMControl

STLeafImpl is included by nothing

StoragePrograms is included by
CachedRegionImplA DiagnosticPilotImpl
FileImpl FilerControl
MStoreImpl PilotControl
ResidentMemoryImpl SimpleSpaceImpl
SnapshotImpl STLeafImpl
SwapBufferImpl SwapperControl
TransactionImpl UtilitySpaceImpl
UtilityVMMControl VMMControl

StoreDriverStartChain is included by
PilotControl SA4000Impl
StartChainPlug

Stream is included by

```
  NetworkStream          NetworkStreamInstance
  NetworkStreamInternal  NetworkStreamMgr
  StreamImpl

StreamImpl is included by nothing

STree is included by
  HierarchyImpl          ProjectionImpl
  STreeImpl              VMMControl

STreeImpl is included by nothing

SubSys is included by
  PilotLoaderSupport     PilotUnLoader

SubVolume is included by
  FileImpl               FilerTransferImpl
  MarkerPageImpl         PhysicalVolumeImpl
  ScavengeImpl           SubVolumeImpl
  VolumeImpl

SubVolumeImpl is included by nothing

SwapBuffer is included by
  CachedRegionImplB      SwapBufferImpl

SwapBufferImpl is included by nothing

SwapperControl is included by nothing

SwapperException is included by
  SpaceImplA             SwapperControl
  SwapperExceptionImpl   UtilitySpaceImpl

SwapperExceptionImpl is included by nothing

SwapperPrograms is included by
  CachedRegionImplB      CachedSpaceImpl
  MStoreImpl             PageFaultImpl
  SwapBufferImpl         SwapperControl
  SwapperExceptionImpl   SwapTaskImpl

SwapTask is included by
  CachedRegionImplB      SwapTaskImpl

SwapTaskImpl is included by nothing

System is included by
  Boot                   BootChannelDisk
  DiagnosticPilotImpl    DiagnosticPilotTest
  EchoServerImpl         EthernetDriver
  EthernetOneDriver      File
  FileCache              FileCacheImpl
  FileImpl               FileInternal
  FilePageLabel          FilerTransferImpl
  FileTaskImpl           FrameImpl
  HalfDuplexImpl         KernelFile
  LogicalVolume          MarkerPage
  MarkerPageImpl         NetworkStream
  NetworkStreamMgr       PacketStreamInstance
  PhysicalVolume         PhysicalVolumeFormat
  PhysicalVolumeImpl     PilotControl
  PilotCounter           PilotDisk
  PilotLoaderSupport     PilotPerfMonitor
  RemotePageTransfer     RouterImpl
  Runtime                SA4000HeadDO
  SA800HeadDO            ScavengeImpl
  Scavenger              SimpleSpaceImpl
  SnapshotImpl           Socket
  SocketImpl             SpaceImplA
  SpaceImplB             SpecialVolume
  STLeafImpl             SubVolume
  SubVolumeImpl          SystemImpl
  TemporaryBooting       TransactionImpl
  TransactionInternal    TransactionLogImpl
  TransactionState       UserTerminalImpl
  UtilitySpaceImpl       VMMControl
  VolAllocMapImpl        VolFileMapImpl
  Volume                 Volumeextras
  VolumeImpl             VolumeImplInterface
  VolumeInternal

System6TablesDO is included by nothing

SystemBufferPoolImpl is included by nothing

SystemImpl is included by nothing

SystemInternal is included by
  DiagnosticPilotImpl    DiagnosticPilotStubsA
  FileImpl               HeapImpl
  PilotControl           SA4000HeadDO
  ScavengeImpl           SpaceImplB
  StreamImpl             SystemImpl
  UtilitySpaceImpl       VolumeInternal

Table is included by
  BcdDefs                BcdOperations
  BcdOps                 PilotLoaderCore
  PilotLoaderOps         PilotLoaderSupport
  PilotUnLoader

Teledebug is included by
  BootSwapGerm           TeledebugImpl

TeledebugImpl is included by nothing

TeledebugProtocol is included by
  TeledebugImpl

TemporaryBooting is included by
  BootSwap               CachedSpaceImpl
  DiagnosticPilotImpl    EthernetDriver
  EthernetOneDriver      FileImpl
  HeapImpl               MapLogImpl
  MStoreImpl             PhysicalVolumeImpl
  PilotControl           PilotNub
  PilotSwitches          SimpleSpaceImpl
  Snapshot               SnapshotImpl
  STLeafImpl             TransactionImpl
  UserTerminalImpl       VolumeImpl

TemporarySetGMT is included by
  SetGMTUsingEthernet    SnapshotImpl
  SystemImpl

TemporaryTransaction is included by
  TransactionImpl

TextBlt is included by
  SoftwareTextBlt        TextBltImpl

TextBltImpl is included by nothing

TimeStamp is included by
  BcdDefs                BcdOperations
  BcdOps                 FrameImpl
  PilotLoaderCore

Transaction is included by
  CachedSpace            DiagnosticPilotImpl
  DiagnosticPilotStubsA  File
  FileImpl               FloppyChannelImpl
  HeapImpl               HierarchyImpl
  KernelFile             KernelSpace
  PilotCommUtil          PilotControl
  PilotCounter           PilotLoaderSupport
  PilotLoadState         SA800Impl
  ScavengeImpl           SimpleSpaceImpl
  SnapshotImpl           Space
  SpaceImplA             SpaceImplB
  STLeafImpl             STreeImpl
  TransactionImpl        TransactionLogImpl
  TransactionState       TransactionStateImpl
  UserTerminalImpl       UtilitySpaceImpl
  UtilityVMMControl

TransactionImpl is included by nothing

TransactionInternal is included by
  TransactionImpl        TransactionLogImpl
  TransactionStateImpl

TransactionLogImpl is included by nothing

TransactionState is included by
  FileImpl               SpaceImplA
  SpaceImplB             TransactionImpl
  TransactionInternal    TransactionLogImpl
  TransactionStateImpl   UtilityVMMControl

TransactionStateImpl is included by nothing

Trap is included by
  BootSwapGerm           PageFaultImpl
  PageFaultTestImpl      SpaceImplB
  Traps                  UtilitySpaceImpl

Traps is included by nothing

TTYPort is included by
  TTYPortDriverA         TTYPortDriverB
  TTYPortInternal

TTYPortDriverA is included by nothing

TTYPortDriverB is included by nothing

TTYPortEnvironment is included by
  TTYPort                TTYPortDriverA
  TTYPortDriverB         TTYPortface
  TTYPortHeadDO          TTYPortInternal

TTYPortface is included by
  TTYPortDriverA         TTYPortDriverB
  TTYPortHeadDO

TTYPortHeadDO is included by nothing

TTYPortInternal is included by
  TTYPortDriverA         TTYPortDriverB

UserTerminal is included by
  UserTerminalImpl

UserTerminalHeadDO is included by nothing

UserTerminalImpl is included by nothing

Utilities is included by
  CachedRegion           CachedRegionImplB
  FileImpl               FilerTransferImpl
  FileTaskImpl           FrameImpl
  MapLogImpl             MarkerPageImpl
  PhysicalVolumeImpl     ResidentMemoryImpl
  SA4000Impl             SA800Impl
  ScavengeImpl           SimpleSpace
  STLeafImpl             STreeImpl
  TeledebugImpl          TransactionImpl
  TransactionLogImpl     UtilitiesImpl
  VolAllocMapImpl        VolFileMapImpl
  VolumeImpl
```

UtilitiesImpl is included by nothing

UtilitySpaceImpl is included by nothing

UtilityVMMControl is included by nothing

VM is included by

CachedRegion	CachedRegionImplA
CachedRegionImplB	CachedRegionInternal
CachedSpace	CachedSpaceImpl
DiagnosticPilotImpl	FileImpl
FilePageTransfer	FileTaskImpl
Hierarchy	HierarchyImpl
MapLog	MapLogImpl
MarkerPageImpl	MStore
MStoreImpl	PageFault
PageFaultImpl	PhysicalVolumeImpl
Projection	ProjectionImpl
ResidentMemoryImpl	ScavengeImpl
SimpleSpace	SimpleSpaceImpl
SpaceImplA	SpaceImplB
SpaceImplInternal	STLeaf
STLeafImpl	STree
STreeImpl	SwapBuffer
SwapBufferImpl	SwapperControl
SwapperException	SwapperExceptionImpl
SwapTask	SwapTaskImpl
TransactionLogImpl	UtilitySpaceImpl
UtilityVMMControl	VMMControl
VMMgrStore	VMMPrograms
VolAllocMapImpl	VolFileMapImpl
VolumeImpl	

VMMapLog is included by

CachedRegionImplB	FileImpl
KernelFile	MapLogImpl
PilotControl	

VMMControl is included by nothing

VMMgrStore is included by

MapLogImpl	SpaceImplB
STLeafImpl	

VMMPrograms is included by

HierarchyImpl	MapLogImpl
ProjectionImpl	SpaceImplA
SpaceImplB	STLeafImpl
UtilitySpaceImpl	UtilityVMMControl
VMMControl	

VolAllocMap is included by

FileImpl	PhysicalVolumeImpl
ScavengeImpl	VolAllocMapImpl
VolFileMapImpl	VolumeImpl

VolAllocMapImpl is included by nothing

VolFileMap is included by

FileImpl	PhysicalVolumeImpl
ScavengeImpl	VolFileMapImpl
VolumeImpl	

VolFileMapImpl is included by nothing

Volume is included by

Boot	DiagnosticPilotImpl
FileCache	FileCacheImpl
FileImpl	FileInternal
FilerTransferImpl	KernelFile
LogicalVolume	MarkerPageImpl
PhysicalVolume	PhysicalVolumeImpl
ScavengeImpl	Scavenger
SnapshotImpl	SpaceImplB
SpecialVolume	STLeafImpl
SubVolume	SubVolumeImpl
SystemImpl	TemporaryBooting
TransactionImpl	TransactionStateImpl
UtilitySpaceImpl	VolAllocMapImpl
VolFileMapImpl	Volumeextras
VolumeImpl	VolumeImplInterface
VolumeInternal	

Volumeextras is included by
- VolumeImpl

VolumeImpl is included by nothing

VolumeImplInterface is included by

PhysicalVolumeImpl	VolumeImpl

VolumeInternal is included by

FileCacheImpl	FileImpl
FileInternal	FilerTransferImpl
LabelTransfer	LogicalVolume
MarkerPageImpl	PhysicalVolumeImpl
ScavengeImpl	SubVolume
SubVolumeImpl	VolAllocMap
VolAllocMapImpl	VolFileMapImpl
VolumeImpl	VolumeImplInterface

X800TablesD0 is included by nothing

XferTrap is included by

PilotControl	PilotCounter
PilotNub	PilotPerfMonitor
Traps	

Zone is included by

DiskChannelImpl	FileCacheImpl
FloppyChannelImpl	HeapImpl
MIOCCommCmdsD0	MIOCCommImplD0
MIOCHardwareD0	MIOCInternalD0
PilotCommUtil	ResidentHeap
ResidentHeapImpl	RS232CHeadFrontEndD0
SA4000Impl	SpaceImplB
STLeafImpl	STreeImpl
SwapBufferImpl	TransactionStateImpl
UserTerminalImpl	ZoneImpl
ZoneInternal	

ZoneImpl is included by nothing

ZoneInternal is included by

HeapImpl	ResidentHeapImpl
ZoneImpl	