DIRECTORY
    AltoDefs: FROM "altodefs" USING [
        CharsPerPage, CharsPerWord, PageCount, PageNumber, PageSize],
    AltoFileDefs: FROM "altofiledefs" USING [eofDA, FA, fillinDA, FP, vDA],
    BFSDefs: FROM "bfsdefs" USING [ActOnPages, GetNextDA, WritePages],
    DiskDefs: FROM "diskdefs" USING [DiskRequest],
    InlineDefs: FROM "inlinedefs" USING [BITAND, BITSHIFT, COPY],
    MiscDefs: FROM "miscdefs" USING [Zero],
    SegmentDefs: FROM "segmentdefs" USING [UpdateFileLength],
    StreamDefs: FROM "streamdefs" USING [
        DiskHandle, StreamErrorCode, StreamHandle],
    SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

DEFINITIONS FROM AltoDefs, AltoFileDefs, StreamDefs;

StreamsA: PROGRAM
    IMPORTS BFSDefs, MiscDefs, SegmentDefs, SystemDefs
    EXPORTS StreamDefs
    SHARES StreamDefs, SegmentDefs
    BEGIN


-- block mode transfers
direction: TYPE = {in,out};
-- the fast stream overflow handler; should only be called
-- from the fast stream get, put, and endof routines. It
-- always supplies a new count (which may be zero, in which
-- case get and/or put is replaced with an error routine).

-- Cleanup makes the disk look like the stream, unless the
-- current page is not full and you didn't ask for a flush.

Fixup: PROCEDURE [stream:StreamHandle] =
    BEGIN pos: CARDINAL;
    WITH s:stream SELECT FROM
        Disk => BEGIN
        Cleanup[s,FALSE]; -- don't flush
        IF (pos + Pos[s]) > s.char THEN BEGIN
            SetEnd[s,TRUE]; -- ran into eof
            Setup[s,pos,CharsPerPage];
        END;
        END;
    ENDCASE => SIGNAL StreamError[s,StreamType];
    RETURN
    END;

Cleanup: PROCEDURE [s:DiskHandle, flush:BOOLEAN] =
    BEGIN pos: CARDINAL;
    IF (pos + Pos[s]) > s.char THEN PositionByte[s,pos,FALSE];
    IF pos=CharsPerPage THEN -- write current page, read (maybe create) next one
        IF s.dirty THEN [] + TransferPages[s,NIL,1,out,FAIL]
        -- donothing with current page, read next one
        ELSE [] + TransferPages[s,NIL,1,in,TRUE]
    ELSE IF s.dirty AND flush THEN
        BEGIN -- write current page w/ new numChars
            Char + TransferPages[s,NIL,0,out,TRUE];
            PositionByte[s,pos,FALSE];
        END;
    END;
    RETURN
    END;

ReadBlock: PUBLIC PROCEDURE [stream:StreamHandle, address:POINTER, words:CARDINAL]
    RETURNS [CARDINAL] =
    BEGIN
done: CARDINAL = 0;
    WITH s:stream SELECT FROM
StreamsA.mesa

2-Sep-78 13:58:17

Disk ➔ IF s.read THEN
  done + TransferBlock[@s,address,words,in];
ENDCASE ➔ [SIGNAL StreamError[@s,StreamType];
RETURN[done]
END;

WriteBlock: PUBLIC PROCEDURE [stream:StreamHandle, address:POINTER, words:CARDINAL] RETURNS [CARDINAL] •
BEGIN
done: CARDINAL + 0;
WITH s:stream SELECT FROM
Disk ➔
  IF (~.s.write AND ~.s.append)
  OR (~.s.write AND .s.append AND ~EndOf[@s])
  OR (.s.write AND ~.s.append AND EndOf[@s])
   THEN NULL
   ELSE done + TransferBlock[@s,address,words,out];
ENDCASE ➔ [SIGNAL StreamError[@s,StreamType];
RETURN[done]
END;

TransferBlock: PROCEDURE [s:DiskHandle, a:POINTER, n:CARDINAL, d:direction] RETURNS [CARDINAL] •
BEGIN
  OPEN InlineDefs;
  np: PageCount;
  done: CARDINAL + 0;
  left, pos, words: CARDINAL;
  IF BITAND[Pos[s],CharsPerWord-1]#0
   THEN ERROR StreamError[s,StreamPosition]:
  WHILE done # n DO
    left + n-done;
    pos + Pos[s]/CharsPerWord;
    words =
      (IF d=out AND .s.append THEN PageSize
       ELSE (.s.char+CharsPerWord-1)/CharsPerWord) - pos;
    words = IF left > words THEN words ELSE left;
    IF words # 0 THEN
      BEGIN
        PositionByte[s,(pos+words)*CharsPerWord,d~in];
        SELECT d FROM
          in ➔ COPY[from:s.buffer.word+pos,to:a,nwords:words];
          out ➔
            COPY[from:a,to:s.buffer.word+pos,nwords:words];
          s.dirty ~ TRUE;
      END;
      ENDCASE;
    END;
    IF s.char # CharsPerPage
    AND s.endof[s] AND (d=in OR ~.s.append)
    THEN RETURN [done+words];
    np + LOOPHOLE[left-words, CARDINAL]/PageSize;
    IF left-words # a THEN
      words + TransferPages[s,a+words,np,d,FALSE]*PageSize + words;
      a + a+words; done + done+words;
    ENDLOOP;
  END;
  RETURN[done]
END;

-- Transfers np pages (or fewer if the file runs out while reading/updating),
-- starting at address a and the current page of the file (the one in
-- the buffer). It leaves the next page in the buffer, with the stream
-- set up at the first character. Note that if writing, the next page
-- is read, not written; if the file is extended, the buffer is cleared.
-- Returns the number of pages transferred, not counting the next one
-- that was read into the buffer. It's only legal to call TransferPages
-- when the buffer is full or empty; use TransferBlock otherwise.

-- Some special uses:
-- a=0 All transfers are into buffer (useful for positioning).
-- np=0 The current page is transferred (useful for Cleanup).
-- np=-1 Backup one page (useful for positioning).

-- The last argument is for very special uses (described below), do
-- not supply it unless you know what you are doing! If special is
-- true, the following funny things happen, depending on direction:
-- direction-in: action is made DoNothing (np should be one)
-- Used by Cleanup to skip the current page and read next one.
-- direction-out: lastAction is replaced by WriteD, and last-
-- Bytes is replaced by the numChars from the stream (np should
-- be zero). Used by Cleanup to flush with new buffer length.

TransferPages: PROCEDURE [
  s:DiskHandle, a:POINTER, np:INTEGER, d:direction, special:BOOLEAN]
RETURNS [PageCount] •
BEGIN
  OPEN DiskDefs:
  backup: BOOLEAN;
  arg: DiskRequest:
  i, fp, lp: PageNumber:
  dobuff: BOOLEAN = FALSE;
  DAs: POINTER TO ARRAY [0 .. 0] OF vDA;
  CAS: POINTER TO ARRAY [0 .. 0] OF POINTER;
  caa: ARRAY [0 .. 4] OF POINTER:
  daa: ARRAY [0 .. 4] OF vDA:
  f: POINTER TO FP = @s.file.fp;
  -- flush the buffer if the transfer won't
  IF d-in THEN
    IF s.dirty THEN Cleanup[s,TRUE]
    ELSE NULL: -- should mark written
    -- include the buffer if the transfer doesn't
    IF a = NIL AND Pos[s] = CharsPerPage THEN
      BEGIN
        -- the stream is at [page n, byte 0], but the
        -- buffer is at [page n-1, byte CharsPerPage]:
        -- transfer the buffer, too, even if not dirty.
        dobuff = TRUE; np = np+1;
        a = a-PageSize; -- fixed below
      END:
      fp = s.page; PositionByte[s,0,d-in];
      IF backup = (np-1) THEN
        BEGIN fp = fp-1; np = 0 END;
      lp = fp+np;
      CAS =
      (IF np = 1 THEN @caa ELSE SystemDefs.AllocateHeapNode[np+3])-(fp-1);
      DAs =
      (IF np = 1 THEN @daa ELSE SystemDefs.AllocateHeapNode[np+3])-(fp-1);
      FOR i IN [fp .. lp] DO
        CAS[i] =
          IF a=NIL THEN s.buffer.word
          ELSE a+(i-fp)*PageSize;
        DAs[i] = fillinDA;
      ENDOLOOP;
      DAs[fp-1] = DAs[lp+1] = fillinDA;
      CAS[lp] = s.buffer.word; IF dobuff THEN CAS[fp] = s.buffer.word;
      InlineDefs.COPY [from:@s.das,to:@DAs[IF backup THEN fp ELSE fp-1],
        nwords:IF backup THEN LENGTH[s.das]-1 ELSE LENGTH[s.das]]:
      arg = DiskRequest [0Cas[0],0das[0].fp,1p,f,FALSE,
        WriteD,ReadD,FALSE,update[BFSDefs.GetNextOA]];
      IF d-in OR (d-out AND ~special AND ~s.append) THEN
        BEGIN
          IF d-in THEN arg.action = ReadD;
          IF special THEN arg.action = DoNothing;
            [i,s.char] = BFSDefsActionButtonPages[LOOPHOLE[@arg]]:
          IF #lp AND s.char>0 AND CAS[i]#s.buffer.word THEN
            InlineDefs.COPY [from:CAS[i],to:s.buffer.word,nwords:PageSize];
          END
        ELSE
          BEGIN
            arg.lastBytes = IF special THEN s.char ELSE 0;
            arg.lastAction = IF special THEN WriteD ELSE ReadD;
            [i,s.char] = BFSDefs.WritePages[LOOPHOLE[@arg]];
          END;
          s.page = i;
          IF s.char=0 THEN MiscDefs.Zero[s.buffer.word,PageSize];
          InlineDefs.COPY [from:0das[i-1].to:0s.das,nwords:LENGTH[s.das]]:
          IF s.das[next]=eofDA THEN
            BEGIN OPEN s:
              fa: FA = FA[das[current],page,char];
StreamsA.mesa

SegmentDefs.UpdateFileLength[file, @fa];
END;
IF np > 1 THEN SystemDefs.FreeHeapNode[CAs+fp-1];
IF np > 1 THEN SystemDefs.FreeHeapNode[DAs+fp-1];
Setup[s, 0, s.char];
SetEnd[s.s.index*s.size]; s.dirty = FALSE;
RETURN[i-fp-(IF dobuffer THEN 1 ELSE 0)];
END;

PositionByte: PROCEDURE [s:DiskHandle, b:CARDINAL, reading: BOOLEAN] =
BEGIN
OPEN s;
pos: CARDINAL;
IF das[next]=eof DA THEN
BEGIN
IF (pos + Pos[s]) > char
AND append AND dirty
THEN char + pos;
IF b > char THEN
IF -append OR reading THEN b + char
ELSE BEGIN char + b; dirty = TRUE END;
END;
Setup[s,b,char];
SetEnd[s,s.index*s.size AND char#CharsPerPage];
RETURN END;

-- FAST STREAMS
-- the counts and positions should be optimized for
-- the instruction set (as in the bcpl implementation).

Setup: PROCEDURE [s:DiskHandle, pos, end:CARDINAL] =
BEGIN
OPEN InlineDefs;
mask: WORD = -s.unit;
shift: INTEGER = s.unit-1;
-- both pos and end are rounded
s.size = BITSHIFT[BITAND[end+LOOPHOLE[shift, CARDINAL].mask],-shift];
s.index = BITSHIFT[BITAND[pos+LOOPHOLE[shift, CARDINAL].mask],-shift];
RETURN END;

Pos: PROCEDURE [s:DiskHandle] RETURNS [CARDINAL] =
BEGIN
RETURN [InlineDefs.BITSHIFT[s.index,s.unit-1]]
END;

SetEnd: PROCEDURE [s:DiskHandle, b:BOOLEAN] =
BEGIN
g: PROCEDURE [StreamHandle] RETURNS [UNSPECIFIED];
p: PROCEDURE [StreamHandle,UNSPECIFIED];
IF s.eof # b THEN
BEGIN
  s.eof + b;
g + s.get; s.get + s.savedGet; s.savedGet + g;
p + s.put; s.put + s.savedPut; s.savedPut + p;
END;
RETURN END;

ReadByte: PROCEDURE [stream:StreamHandle] RETURNS [item:UNSPECIFIED] =
BEGIN
WITH s:stream SELECT FROM
Disk =>
BEGIN
IF s.index = s.size THEN
BEGIN
  s.getOverFlow[@s];
  RETURN[s.get[@s]]; END;
item + s.buffer.byte[s.index];
s.index + s.index + 1;
END;
ENDCASE =>
BEGIN
SIGNAL StreamError[@s.StreamType];
item + 0; END;
RETURN END;

ReadWord: PROCEDURE [stream:StreamHandle] RETURNS [item:UNSPECIFIED] =
BEGIN
WITH s:stream SELECT FROM
Disk =>
BEGIN
IF s.index = s.size THEN
BEGIN s.getOverflow[@s]; RETURN[s.get[@s]]; END;
item = s.buffer.word[s.index];
s.index = s.index + 1;
END;
ENDCASE =>
BEGIN SIGNAL StreamError[@s,StreamType]; item = 0; END;
RETURN END;

WriteByte: PROCEDURE [stream:StreamHandle, item:UNSPECIFIED] =
BEGIN
WITH s:stream SELECT FROM
Disk =>
BEGIN
IF s.index = s.size THEN
BEGIN s.putOverflow[@s]; s.put[@s,item]; RETURN; END;
s.buffer.byte[s.index] = item;
s.index = s.index + 1;
s.dirty = TRUE;
END;
ENDCASE => SIGNAL StreamError[@s,StreamType];
RETURN END;

WriteWord: PROCEDURE [stream:StreamHandle, item:UNSPECIFIED] =
BEGIN
WITH s:stream SELECT FROM
Disk =>
BEGIN
IF s.index = s.size THEN
BEGIN s.putOverflow[@s]; s.put[@s,item]; RETURN; END;
s.buffer.word[s.index] = item;
s.index = s.index + 1;
s.dirty = TRUE;
END;
ENDCASE => SIGNAL StreamError[@s,StreamType];
RETURN END;

EndOf: PROCEDURE [stream:StreamHandle] RETURNS [BOOLEAN] =
BEGIN
WITH s:stream SELECT FROM
Disk =>
BEGIN
IF s.eof THEN RETURN[TRUE];
IF s.index#s.size THEN RETURN[FALSE];
s.getOverflow[@s]; RETURN[s.endof[@s]];
END;
ENDCASE => SIGNAL StreamError[@s,StreamType];
RETURN[FALSE]
END;

END.