-- Keyboard.mesa Edited by Sandman on May 12, 1978 9:20 AM

DIRECTORY
  ControlDefs: FROM "controldefs" USING [SD, StateVector],
  CoreSwapDefs: FROM "coreswapdefs" USING [PuntInfo],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITSHIFT, BITXOR],
  KeyDefs: FROM "keydefs" USING [KeyArray, KeyBits, KeyItem, updown],
  Mopcodes: FROM "mopcodes" USING [zBRK, zKFCB],
  ProcessDefs: FROM "processdefs" USING [Priority],
  SDDefs: FROM "sddefs" USING [sFirstStateVector, sInterrupt],
  StreamDefs: FROM "streamdefs" USING [
    KeyboardHandle, KeyBufChars, StreamHandle];

DEFINITIONS FROM ProcessDefs, InlineDefs, KeyDefs, StreamDefs;

Keyboard: MONITOR LOCKS monitor
  EXPORTS KeyDefs SHARES StreamDefs =

BEGIN

monitor: PUBLIC MONITORLOCK;
wakeup: PUBLIC CONDITION;
charactersAvailable: PUBLIC CONDITION;

-- variables set by KeyStreams
ks: PUBLIC KeyboardHandle;

COT: PUBLIC BOOLEAN;
cursorTracking: PUBLIC BOOLEAN;

KeyTable: PUBLIC POINTER TO ARRAY [0..80] OF KeyItem;

-- The Keyboard part:

-- fixed addresses for keyboard and mouse
Keys: PUBLIC POINTER TO KeyArray;

Coordinate: TYPE = RECORD [x,y: INTEGER];
Mouse: PUBLIC POINTER TO Coordinate;
Cursor: PUBLIC POINTER TO Coordinate;
Xmax: CARDINAL = 606-16;
Ymax: CARDINAL = 808-16;

ns, os: KeyArray;
OldState: PUBLIC POINTER TO KeyArray = @os;
NewState: POINTER TO KeyArray = @ns;

KeyboardPriority: PUBLIC Priority = 6;
GetDebugger: PROCEDURE =
  MACHINE CODE BEGIN Mopcodes.zKFCB, SDDefs.sInterrupt END;

ProcessKeyboard: PUBLIC ENTRY PROCEDURE =
BEGIN
  bitcount, start: [0..15];
  char: [0..377B];
  entry: KeyItem;
i: [0..SIZE[KeyArray]];
interruptState: updown + up;
newin: CARDINAL;
pp: Priority;
StateWord: WORD;
SV: POINTER TO ARRAY Priority OF ControlDefs.StateVector =
  ControlDefs.DO[SDDefs.sFirstStateVector];
stroke: POINTER TO KeyBits = LOOPHOLE[NewState];

DO
  WAIT wakeup;
  -- first update the cursor
  IF cursorTracking THEN
    BEGIN
      Mouse.x += Cursor.x + MAX[MIN[Xmax,Mouse.x], 0];
      Mouse.y += Cursor.y + MAX[MIN[Ymax,Mouse.y], 0];
    END;
  
NewState? + Keys;
-- The following code checks for Ctrl-Swat, the debugger interrupt keys.
-- This code could be made into a separate process.
IF stroke.Ctrl • down AND stroke.Spare3 • down THEN
BEGIN
IF interruptState • up AND CoreSwapDefs.PuntInfo+ # LOOPHOLE[0] THEN
BEGIN
interruptState • down;
FOR pp IN [0..KeyboardPriority) DO
  SV[pp].instbyte • Mopcodes.zBRK;
WAIT wakeup;
IF SV[pp].instbyte • 0 THEN EXIT
ELSE SV[pp].instbyte • 0;
REPEAT FINISHED arrows GetDebugger[];
ENDLOOP;
NewState • Keys;
END;
ELSE interruptState • up;
-- The following code checks for down transitions in the keyboard state
-- and enters characters in the current keystream buffer
FOR i IN [0..SIZE[KeyArray]) DO
IF (StateWord • BITXOR[OldState[i],NewState[i]]) # 0 THEN
BEGIN -- found one or more transitions
  start • 0;
  DO
    FOR bitcount IN [start .. 15] DO
      IF LOOPHOLE[StateWord,INTEGER]<0 THEN EXIT;
    StateWord • BITSHIFT[StateWord,1];
    ENDLOOP;
    entry • KeyTable[i*16 + bitcount];
    IF (char + entry.NormalCode) # 0
AND BITAND[01dState[i],BITSHIFT[100000B,-bitcount]] # 0 THEN
BEGIN
  SELECT updown[down] FROM stroke.Ctrl =)
  IF char • 177B THEN BEGIN
    COT • TRUE;
    GOTO skip END
ELSE char • BITAND[char, 37B];
  stroke.LeftShift, stroke.RightShift =>
  char • entry.ShiftCode;
  stroke.Lock =>
  IF entry.Letter THEN char • entry.ShiftCode;
  ENDCASE;
  IF (newin~ks.in+l) = KeyBufChars THEN newin • 0;
  IF newin # ks.out THEN
BEGIN
  ks.buffer[ks.in] • LOOPHOLE[char];
  ks.in • newin;
  BROADCAST charactersAvailable;
  END;
  EXITS skip => NULL;
  END;
IF (StateWord • BITSHIFT[StateWord,1])=0 THEN EXIT;
start • bitcount+1;
ENDLOOP;
END;
ENDLOOP;
OldState • NewState;
ENDLOOP;
END;

ReadChar: PUBLIC ENTRY PROCEDURE [stream: StreamHandle]
RETURNS [char: UNSPECIFIED] =
BEGIN
char • 0;
WITH s:stream SELECT FROM
Keyboard =>
DO -- until character typed
IF s.out # s.in THEN
BEGIN
  char • s.buffer[s.out];
s.out •
  IF s.out = KeyBufChars-1
  THEN 0
  ELSE s.out+1;
  EXIT;
END;
RETURN
END;
WAIT charactersAvailable;
ENDLOOP;
ENDCASE;
RETURN;
END;

InputBufferEmpty: PUBLIC ENTRY PROCEDURE [stream:StreamHandle]
RETURNS [empty: BOOLEAN] =
BEGIN
empty ← TRUE;
WITH s:stream SELECT FROM
  Keyboard → IF s.in ≠ s.out THEN empty ← FALSE;
ENDCASE;
RETURN
END;

OldState† ← Keys†;
END.