```
-- Signaller.Mesa  Edited by Sandman on May 12, 1978  3:09 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BYTE],
  ControlDefs: FROM "controldefs" USING [
    Alloc, ControlLink, Frame, FrameHandle, Free, GetReturnFrame,
    GetReturnLink, Greg, InstWord, localbase, Lreg, NullFrame, SetReturnLink,
    StateVector, WordPC],
  FrameDefs: FROM "framedefs" USING [SwapInCode],
  ImageDefs: FROM "imagedefs" USING [PuntMesa],
  Mopcodes: FROM "mopcodes" USING [
    zCATCH, zJ2, zJ9, zJB, zJW, zKFCB, zNOOP, zPORTI, zSLB],
  NucleusDefs: FROM "nucleusdefs",
  SDDefs: FROM "sddefs" USING [
    SD, sError, sErrorList, sReturnError, sReturnErrorList, sSignal,
    sSignalList, sUncaughtSignal, sUnnamedError],
  SegmentDefs: FROM "segmentdefs" USING [Unlock],
  TrapDefs: FROM "trapdefs";

DEFINITIONS FROM ControlDefs;

Signaller: PROGRAM
  IMPORTS FrameDefs, SegmentDefs
  EXPORTS FrameDefs, NucleusDefs, TrapDefs =
  BEGIN

  BYTE: TYPE = AltoDefs.BYTE;

  CatchPointer: TYPE = POINTER TO catch Frame;
  CatchCall: TYPE = PROCEDURE [SIGNAL] RETURNS [ActionCode];
  CatchContinue: TYPE = PROCEDURE;

  ActionCode: TYPE = INTEGER;
    reject: ActionCode = 0;
    resume: ActionCode = 1;
    exit:   ActionCode = -1;

  SendMsgSignal: PUBLIC SIGNAL RETURNS [UNSPECIFIED, UNSPECIFIED] = CODE;

  signalling: CARDINAL = 177777B;
  notSignalling: CARDINAL = 0;

  MarkSignalFrame: PROCEDURE [value: CARDINAL] =
    MACHINE CODE BEGIN Mopcodes.zSLB, 3 --OFFSET[mark]-- END;


  SignalHandler: PROCEDURE [signal: SIGNAL, message: UNSPECIFIED] =
    BEGIN

    SignalFrame: TYPE = POINTER TO FRAME[SignalHandler];

    frame, nextFrame: FrameHandle;
    target, nextTarget: FrameHandle;
    self: FrameHandle = REGISTER[Lreg];
    start: FrameHandle;
    catchFrame: CatchPointer;
    action: ActionCode;
    unwinding: BOOLEAN;
    catchPhrase: BOOLEAN;
    catchFSIndex: BYTE;
    catchPC, exitPC: WordPC;
    catchState: ControlDefs.StateVector;

    MarkSignalFrame[signalling];  unwinding ← FALSE;
    start ← GetFrame[self.returnlink];  target ← NullFrame;
      DO
      nextFrame ← start;
      UNTIL nextFrame = target DO
        frame ← nextFrame;
        IF frame.accesslink = REGISTER[Greg] AND frame.mark THEN
          BEGIN
          OPEN thisSignaller: LOOPHOLE[frame, SignalFrame];
          IF unwinding THEN
            BEGIN
            IF signal = thisSignaller.signal THEN
              nextTarget ← IF thisSignaller.unwinding
```

```
            THEN thisSignaller.nextTarget
            ELSE thisSignaller.nextFrame;
          IF thisSignaller.unwinding THEN
            BEGIN
            IF thisSignaller.frame = LOOPHOLE[frame.returnlink] THEN
              frame.returnlink ← [frame[thisSignaller.nextFrame]];
            ControlDefs.Free[thisSignaller.frame];
            END;
          nextFrame ← GetFrame[frame.returnlink];
          END
        ELSE
          nextFrame ← IF signal # thisSignaller.signal
            THEN
              IF thisSignaller.unwinding
                THEN thisSignaller.nextFrame
                ELSE GetFrame[frame.returnlink]
            ELSE
              IF thisSignaller.unwinding
                THEN thisSignaller.nextTarget
                ELSE thisSignaller.nextFrame;
        END
      ELSE nextFrame ← GetFrame[frame.returnlink];
      IF unwinding AND nextTarget = frame THEN  nextTarget ← nextFrame;
      [catchPhrase, catchFSIndex, catchPC] ← CheckCatch[frame];
      IF catchPhrase
        THEN
          BEGIN
          catchFrame ← ControlDefs.Alloc[catchFSIndex];
          catchFrame↑ ← Frame[
              accesslink: frame.accesslink,
              pc: catchPC,
              returnlink: [frame[self]],
              extensions: catch[unused: ,
                staticlink: frame+localbase, messageval: message]];
          action ← LOOPHOLE[catchFrame, CatchCall]
           [IF unwinding THEN LOOPHOLE[UNWIND] ELSE signal
            ! SendMsgSignal => RESUME[message, signal]];
          catchState ← STATE;
          SELECT action FROM
            reject =>  NULL;
            resume =>
              IF unwinding
                THEN ERROR ResumeError
                ELSE
                  BEGIN
                  catchState.dest ← ControlDefs.GetReturnLink[];
                  catchState.source ← 0;
                  RETURN WITH catchState;
                  END;
            exit =>
              BEGIN
              -- catchFrame is waiting to execute its exit jump
              exitPC ← catchFrame.pc;
              ControlDefs.Free[catchFrame];
              target ← LOOPHOLE[catchState.stk[0]-ControlDefs.localbase];
              nextTarget ← nextFrame;
              unwinding ← TRUE;  message ← NIL;
              GO TO StartUnwind;
              END;
            ENDCASE;
          END;
      IF unwinding
        THEN
          BEGIN
          IF frame = start THEN start ← nextFrame;
          IF frame = LOOPHOLE[self.returnlink]
            THEN self.returnlink ← [frame[nextFrame]];
          ControlDefs.Free[frame];
          END;
      REPEAT
        StartUnwind =>  NULL;
        FINISHED =>  EXIT
      ENDLOOP;
    REPEAT
      ucs => NULL;
    ENDLOOP;
```

```
    IF unwinding THEN target.pc ← exitPC
    ELSE
      BEGIN
      IF SDDefs.SD[SDDefs.sUncaughtSignal] = 0 THEN ImageDefs.PuntMesa[];
      UncaughtSignal[message, signal, frame];
      END;
    RETURN
    END;

  CheckCatch: PROCEDURE [frame: FrameHandle]
    RETURNS [catchPhrase: BOOLEAN, fsIndex: BYTE, pc: WordPC] =
    BEGIN  OPEN Mopcodes;
    iw: POINTER TO InstWord;
    parity: {even, odd};
    MarkSignalFrame[notSignalling];
    FrameDefs.SwapInCode[frame.accesslink];
    iw ← ABS[frame.pc] + frame.accesslink.code.codebase;
    parity ← IF frame.pc < 0 THEN odd ELSE even;
    DO
      SELECT (IF parity=even THEN iw.evenbyte ELSE iw.oddbyte) FROM
        zCATCH => BEGIN catchPhrase ← parity = even; EXIT END;
        zPORTI,zNOOP =>
          IF parity=even THEN  parity ← odd
          ELSE  BEGIN  iw ← iw + 1;  parity ← even;  END;
        ENDCASE => BEGIN catchPhrase ← FALSE; EXIT END;
      ENDLOOP;
    IF catchPhrase
      THEN
        BEGIN  -- [iw, parity] points at zCatch (note: parity must be even)
        pc ← WordPC[iw - frame.accesslink.code.codebase + 1];
        fsIndex ← iw.oddbyte;
        SELECT (iw+1).evenbyte FROM
          zJB =>
            BEGIN pc ← [pc + 1]; GO TO evenPC END;
          IN [zJ2..zJ9] =>
            GO TO oddPC;
          zJW => -- always padded !
            BEGIN pc ← [pc + 2];  GO TO evenPC  END;
          ENDCASE;
        EXITS
          evenPC => NULL;
          oddPC => pc ← [-pc];
        END;
    SegmentDefs.Unlock[frame.accesslink.codesegment];
    RETURN
    END;

  GetFrame: PROCEDURE [link: ControlLink] RETURNS [FrameHandle] =
    BEGIN
    -- MarkSignalFrame[notSignalling];
    DO
      WITH cl: link SELECT link.tag FROM
        frame =>  RETURN [cl.frame];
        indirect =>  link ← cl.link↑;
        ENDCASE => RETURN[NullFrame];
      ENDLOOP;
    END;


  Signal: PROCEDURE [signal: SIGNAL, message: UNSPECIFIED] = SignalHandler;

  SignalList: PROCEDURE [signal: SIGNAL, message: POINTER TO UNSPECIFIED] =
    BEGIN
    MarkSignalFrame[notSignalling];
    SignalHandler[signal, message !
      UNWIND => ControlDefs.Free[message]];
    ControlDefs.Free[message];
    RETURN
    END;


  ResumeError: PUBLIC SIGNAL = CODE;

  Error: PROCEDURE [signal: SIGNAL, message: UNSPECIFIED] =
    BEGIN
    MarkSignalFrame[notSignalling];
```

```
    SignalHandler[signal, message];
    ERROR ResumeError
    END;

  ErrorList: PROCEDURE [signal: SIGNAL, message: POINTER TO UNSPECIFIED] =
    BEGIN
    MarkSignalFrame[notSignalling];
    SignalHandler[signal, message !
      UNWIND => ControlDefs.Free[message]];
    ControlDefs.Free[message];
    ERROR ResumeError
    END;

  ReturnError: PROCEDURE [signal: SIGNAL, message: UNSPECIFIED] =
    BEGIN
    caller: FrameHandle = ControlDefs.GetReturnFrame[];
    ControlDefs.SetReturnLink[caller.returnlink];
    MarkSignalFrame[notSignalling];
    SignalHandler[signal, message !
      UNWIND => ControlDefs.Free[caller]];
    ControlDefs.Free[caller];
    ERROR ResumeError
    END;

  ReturnErrorList: PROCEDURE [signal: SIGNAL, message: POINTER TO UNSPECIFIED] =
    BEGIN
    caller: FrameHandle = ControlDefs.GetReturnFrame[];
    ControlDefs.SetReturnLink[caller.returnlink];
    MarkSignalFrame[notSignalling];
    SignalHandler[signal, message !
      UNWIND =>
        BEGIN
        ControlDefs.Free[caller];
        ControlDefs.Free[message];
        END];
    ControlDefs.Free[caller];
    ControlDefs.Free[message];
    ERROR ResumeError
    END;

  UnnamedError: PROCEDURE =
    BEGIN
    MarkSignalFrame[notSignalling];
    SignalHandler[LOOPHOLE[-1], -1];
    ERROR ResumeError
    END;

  UncaughtSignal: PROCEDURE [msg, signal: UNSPECIFIED, frame: FrameHandle] =
    MACHINE CODE BEGIN Mopcodes.zKFCB, SDDefs.sUncaughtSignal END;


  Init: PROCEDURE =
    BEGIN OPEN SDDefs;
    sd: POINTER TO ARRAY [0..0) OF UNSPECIFIED ← SD;
    sd[sSignalList] ← SignalList;
    sd[sSignal] ← Signal;
    sd[sErrorList] ← ErrorList;
    sd[sError] ← Error;
    sd[sReturnErrorList] ← ReturnErrorList;
    sd[sReturnError] ← ReturnError;
    sd[sUnnamedError] ← UnnamedError;
    END;

  Init[];

  END.
```