```
-- LoaderBcdUtilities.mesa
-- Last Modified by Sandman,  May 12, 1978  2:35 PM

DIRECTORY
  BcdDefs: FROM "bcddefs" USING [
    CTHandle, CTIndex, CTNull, CTRecord, EXPHandle, EXPIndex, EXPNull,
    EXPRecord, FTHandle, FTIndex, FTNull, FTRecord, IMPHandle, IMPIndex,
    IMPNull, IMPRecord, MTHandle, MTIndex, MTNull, MTRecord, Namee,
    NameRecord, NTHandle, NTIndex, NTNull, NTRecord, SGHandle, SGIndex,
    SGNull, SGRecord],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [BcdBase],
  LoadStateDefs: FROM "loadstatedefs" USING [UpdateLoadStateDA],
  SegmentDefs: FROM "segmentdefs" USING [
    DeleteFileSegment, FileSegmentAddress, FileSegmentHandle, SwapIn, Unlock];

DEFINITIONS FROM LoaderBcdUtilDefs, BcdDefs;

LoaderBcdUtilities: PROGRAM IMPORTS LoadStateDefs, SegmentDefs EXPORTS LoaderBcdUtilDefs = PUBLIC

  BEGIN


  EnumerateConfigTable: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [CTHandle, CTIndex] RETURNS [BOOLEAN]]
    RETURNS [cth: CTHandle, cti: CTIndex] =
    BEGIN
    ctb: CARDINAL = LOOPHOLE[bcd + bcd.ctOffset];
    FOR cti ← FIRST[CTIndex], cti + SIZE[CTRecord] UNTIL cti = bcd.ctLimit DO
      cth ← ctb+cti;
      IF proc[cth, cti] THEN RETURN;
      ENDLOOP;
    RETURN[NIL, CTNull];
    END;

  EnumerateExportTable: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [EXPHandle, EXPIndex] RETURNS [BOOLEAN]]
    RETURNS [eth: EXPHandle, eti: EXPIndex] =
    BEGIN
    etb: CARDINAL = LOOPHOLE[bcd + bcd.expOffset];
    FOR eti ← FIRST[EXPIndex], eti + SIZE[EXPRecord] + (etb+eti).size
    UNTIL eti = bcd.expLimit DO
      eth ← etb+eti;
      IF proc[eth, eti] THEN RETURN;
      ENDLOOP;
    RETURN[NIL, EXPNull];
    END;

  EnumerateImportTable: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [IMPHandle, IMPIndex] RETURNS [BOOLEAN]]
    RETURNS [ith: IMPHandle, iti: IMPIndex] =
    BEGIN
    itb: CARDINAL = LOOPHOLE[bcd + bcd.impOffset];
    FOR iti ← FIRST[IMPIndex], iti + SIZE[IMPRecord] UNTIL iti = bcd.impLimit DO
      ith ← itb+iti;
      IF proc[itb+iti, iti] THEN RETURN;
      ENDLOOP;
    RETURN[NIL, IMPNull];
    END;

  EnumerateModuleTable: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [MTHandle, MTIndex] RETURNS [BOOLEAN]]
    RETURNS [mth: MTHandle, mti: MTIndex] =
    BEGIN
    mtb: CARDINAL = LOOPHOLE[bcd + bcd.mtOffset];
    FOR mti ← FIRST[MTIndex], mti + SIZE[MTRecord] + (mtb+mti).frame.length
    UNTIL mti = bcd.mtLimit DO
      mth ← mtb+mti;
      IF proc[mth, mti] THEN RETURN;
      ENDLOOP;
    RETURN[NIL, MTNull];
    END;

  EnumerateNameTable: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [NTHandle, NTIndex] RETURNS [BOOLEAN]]
    RETURNS [nth: NTHandle, nti: NTIndex] =
    BEGIN
```

```
    ntb: CARDINAL = LOOPHOLE[bcd + bcd.ntOffset];
    FOR nti ← FIRST[NTIndex], nti + SIZE[NTRecord] UNTIL nti = bcd.ntLimit DO
      nth ← ntb+nti;
      IF proc[nth, nti] THEN RETURN;
      ENDLOOP;
    RETURN[NIL, NTNull];
    END;

  EnumerateSegTable: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [SGHandle, SGIndex] RETURNS [BOOLEAN]]
    RETURNS [sgh: SGHandle, sgi: SGIndex] =
    BEGIN
    sgb: CARDINAL = LOOPHOLE[bcd + bcd.sgOffset];
    FOR sgi ← FIRST[SGIndex], sgi + SIZE[SGRecord] UNTIL sgi = bcd.sgLimit DO
      sgh ← sgb+sgi;
      IF proc[sgh, sgi] THEN RETURN;
      ENDLOOP;
    RETURN[NIL, SGNull];
    END;

  EnumerateFileTable: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [FTHandle, FTIndex] RETURNS [BOOLEAN]]
    RETURNS [fth: FTHandle, fti: FTIndex] =
    BEGIN
    ftb: CARDINAL = LOOPHOLE[bcd + bcd.ftOffset];
    FOR fti ← FIRST[FTIndex], fti + SIZE[FTRecord] UNTIL fti = bcd.ftLimit DO
      fth ← ftb+fti;
      IF proc[fth, fti] THEN RETURN;
      ENDLOOP;
    RETURN[NIL, FTNull];
    END;

  FindName: PROCEDURE [bcd: BcdBase, owner: Namee] RETURNS [name: NameRecord] =
    BEGIN
    n: NTHandle;
    FindOwner: PROCEDURE [nth: NTHandle, nti: NTIndex] RETURNS [BOOLEAN] =
      BEGIN
      RETURN[owner = nth.item];
      END;
    IF (n ← EnumerateNameTable[bcd, FindOwner].nth) = NIL
      THEN name ← [0]
      ELSE name ← n.name;
    RETURN
    END;

  SetUpBcd: PROCEDURE [bcdseg: SegmentDefs.FileSegmentHandle] RETURNS [bcd: BcdBase] =
    BEGIN OPEN SegmentDefs;
    SwapIn[bcdseg];
    RETURN [FileSegmentAddress[bcdseg]];
    END;

  ReleaseBcdSeg: PROCEDURE [bcdseg: SegmentDefs.FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    LoadStateDefs.UpdateLoadStateDA[bcdseg];
    Unlock[bcdseg];
    IF bcdseg.lock = 0 THEN DeleteFileSegment[bcdseg];
    END;


END....
```