```
-- BootUser.Mesa  Edited by Sandman on August 14, 1978  9:47 AM

DIRECTORY
  BootCacheDefs: FROM "bootcachedefs",
  BootmesaDefs: FROM "bootmesadefs",
  ControlDefs: FROM "controldefs",
  FakeSegDefs: FROM "fakesegdefs",
  IODefs: FROM "iodefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs";

DEFINITIONS FROM BootmesaDefs;

BootUser: PROGRAM
  IMPORTS BootmesaDefs, IODefs, BootCacheDefs, StreamDefs, StringDefs, SystemDefs
  EXPORTS BootmesaDefs = PUBLIC
  BEGIN

  NUL: CHARACTER = IODefs.NUL;
  CR: CHARACTER = IODefs.CR;

  OpenSource: PUBLIC PROCEDURE [root: STRING] =
    BEGIN
    name: STRING ← [40];
    i: CARDINAL;
    FOR i IN [0..root.length) DO
      IF root[i] = '. THEN EXIT;
      StringDefs.AppendChar[name, root[i]];
      ENDLOOP;
    StringDefs.AppendString[name, ".bootmesa."L];
    source ← StreamDefs.NewByteStream[name, StreamDefs.Read];
    END;

  source: StreamDefs.StreamHandle;

  GetToken: PROCEDURE [token: STRING] RETURNS [term: CHARACTER] =
    BEGIN OPEN IODefs;
    token.length ← 0;
    DO
      SELECT term ← source.get[source
        ! StreamDefs.StreamError => BEGIN term ← NUL; EXIT END] FROM
        '- =>
          UNTIL source.get[source] = CR DO NULL ENDLOOP;
        SP => NULL;
        CR => IF token.length # 0 THEN BEGIN term ← ';; RETURN END;
        ':, ';, '>, ', => RETURN;
        ENDCASE => StringDefs.AppendChar[token, term];
      ENDLOOP;
    END;

  module: STRING ← [40];
  config: STRING ← [40];
  keyword: STRING ← [40];

  GetModule: PROCEDURE [module, config: STRING] RETURNS [term: CHARACTER] =
    BEGIN
    module.length ← config.length ← 0;
    term ← GetToken[module];
    IF term = ': THEN
      BEGIN OPEN IODefs;
      WriteString["Syntax Error. Expected module Found "L];
      WriteString[module];
      WriteChar[term];
      SIGNAL BootAbort;
      END;
    IF term = '> THEN
      BEGIN
      StringDefs.AppendString[config, module];
      term ← GetToken[module];
      IF term = ': THEN
        BEGIN OPEN IODefs;
        WriteString["Syntax Error. Expected module Found "L];
        WriteString[module];
        WriteChar[term];
        SIGNAL BootAbort;
```

```
      END;
    IF term = '> THEN BootmesaDefs.BootmesaModuleError[
      "Syntax Error. Only one config name allowed "L, config, module];
    END;
  RETURN
  END;

GetNumber: PROCEDURE RETURNS [term: CHARACTER, n: CARDINAL] =
  BEGIN
  token: STRING ← [10];
  null: STRING ← [1];
  term ← GetToken[token];
  n ← StringDefs.StringToDecimal[token !
    StringDefs.InvalidNumber =>
      BootmesaModuleError["Invalid Number"L, null, token]];
  RETURN
  END;

BootmesaModuleError: PUBLIC PROCEDURE [msg, config, module: STRING] =
  BEGIN OPEN IODefs;
  WriteString[msg];
  IF config.length # 0 THEN
    BEGIN WriteString[config]; WriteChar['>]; END;
  WriteString[module];
  SIGNAL BootAbort;
  RETURN
  END;

GetKeyWord: PROCEDURE [keyword: STRING] RETURNS [term: CHARACTER] =
  BEGIN
  keyword.length ← 0;
  term ← GetToken[keyword];
  IF term # ': AND keyword.length # 0 THEN
    BootmesaDefs.BootmesaError["Syntax Error. Expected KEYWORD:"L];
  RETURN
  END;

ParseInput: PUBLIC PROCEDURE =
  BEGIN OPEN StringDefs;
  term: CHARACTER;
  term ← GetKeyWord[keyword];
  UNTIL term = NUL DO
    IF EqualString[keyword, "CONTROL"L] THEN
      BEGIN
      term ← GetModule[module, config];
      AddControl[module, config];
      END
    ELSE IF EqualString[keyword, "NUB"L] THEN
      BEGIN
      term ← GetModule[module, config];
      AddNub[module, config];
      END
    ELSE IF EqualString[keyword, "WART"L] THEN
      BEGIN
      term ← GetModule[module, config];
      AddWart[module, config];
      END
    ELSE IF EqualString[keyword, "RESIDENT"L] THEN
      BEGIN
      DO
        SELECT term ← GetModule[module, config] FROM
          '; => EXIT;
          ', => AddResident[module, config];
          ENDCASE =>
            BootmesaModuleError["Syntax Error. Invalid modulelist at "L,
              config, module];
        ENDLOOP;
      AddResident[module, config];
      END
    ELSE IF EqualString[keyword, "SWAPPEDIN"L] THEN
      BEGIN
      DO
        SELECT term ← GetModule[module, config] FROM
          '; => EXIT;
          ', => AddSwappedIn[module, config];
          ENDCASE =>
```

```
            BootmesaModuleError["Syntax Error. Invalid modulelist at "L,
              config, module];
        ENDLOOP;
      AddSwappedIn[module, config];
      END
    ELSE IF EqualString[keyword, "NOTRAP"L] THEN
      BEGIN
      DO
        SELECT term ← GetModule[module, config] FROM
          '; => EXIT;
          ', => AddNoTrap[module, config];
          ENDCASE =>
            BootmesaModuleError["Syntax Error. Invalid modulelist at "L,
              config, module];
        ENDLOOP;
      AddNoTrap[module, config];
      END
    ELSE IF EqualString[keyword, "GFT"L] THEN
      BEGIN
      length: CARDINAL;
      [term, length] ← GetNumber[];
      IF term # '; THEN BootmesaError["Error. Invalid GFT Length"L];
      BootmesaDefs.SetDefaultGFTLength[length];
      END
    ELSE IF EqualString[keyword, "MEMORY"L] THEN
      BEGIN
      fp, lp: CARDINAL;
      [term, fp] ← GetNumber[];
      IF term # ', THEN BootmesaError["Error. Invalid Memory Bounds"L];
      [term, lp] ← GetNumber[];
      IF term # '; THEN BootmesaError["Error. Invalid Memory Bounds"L];
      BootmesaDefs.SetDefaultMemoryLimits[fp,lp];
      END
    ELSE IF EqualString[keyword, "PROCESSES"L] THEN
      BEGIN
      number: CARDINAL;
      [term, number] ← GetNumber[];
      IF term # '; THEN BootmesaError["Error. Invalid Process Number"L];
      BootmesaDefs.SetDefaultNProcesses[number];
      END
    ELSE IF EqualString[keyword, "CACHE"L] THEN
      BEGIN
      number: CARDINAL;
      [term, number] ← GetNumber[];
      IF term # '; THEN BootmesaError["Error. Invalid Cache Size"L];
      BootCacheDefs.SetDefaultCacheSize[number];
      END
    ELSE
      BEGIN OPEN IODefs;
      WriteString["Syntax Error. Expected Keyword: Found "L];
      WriteString[keyword];
      WriteChar[term];
      SIGNAL BootAbort;
      END;
    term ← GetKeyWord[keyword];
    ENDLOOP;
  RETURN;
  END;

ModuleObject: TYPE = RECORD [
  link: ModuleHandle,
  frame: ControlDefs.GlobalFrameHandle,
  module, config: STRING];

ModuleHandle: TYPE = POINTER TO ModuleObject;

Control, Resident, SwappedIn, Wart, Nub, NoTrap: ModuleHandle ← NIL;
lastResident, lastSwappedIn, lastNoTrap: ModuleHandle ← NIL;

AddControl: PROCEDURE [module, config: STRING] =
  BEGIN OPEN SystemDefs;
  Control ← AllocateHeapNode[SIZE[ModuleObject]];
  Control↑ ← [link: NIL, frame: ControlDefs.NullGlobalFrame,
    module: AllocateHeapString[module.length],
    config: AllocateHeapString[config.length]];
  StringDefs.AppendString[Control.module, module];
```

```
    StringDefs.AppendString[Control.config, config];
    END;

  AddWart: PROCEDURE [module, config: STRING] =
    BEGIN OPEN SystemDefs;
    Wart ← AllocateHeapNode[SIZE[ModuleObject]];
    Wart↑ ← [link: NIL, frame: ControlDefs.NullGlobalFrame,
      module: AllocateHeapString[module.length],
      config: AllocateHeapString[config.length]];
    StringDefs.AppendString[Wart.module, module];
    StringDefs.AppendString[Wart.config, config];
    END;

  AddNub: PROCEDURE [module, config: STRING] =
    BEGIN OPEN SystemDefs;
    Nub ← AllocateHeapNode[SIZE[ModuleObject]];
    Nub↑ ← [link: NIL, frame: ControlDefs.NullGlobalFrame,
      module: AllocateHeapString[module.length],
      config: AllocateHeapString[config.length]];
    StringDefs.AppendString[Nub.module, module];
    StringDefs.AppendString[Nub.config, config];
    END;

  AddResident: PROCEDURE [module, config: STRING] =
    BEGIN OPEN SystemDefs;
    m: ModuleHandle;
    m ← AllocateHeapNode[SIZE[ModuleObject]];
    m↑ ← [link: NIL, frame: ControlDefs.NullGlobalFrame,
      module: AllocateHeapString[module.length],
      config: AllocateHeapString[config.length]];
    StringDefs.AppendString[m.module, module];
    StringDefs.AppendString[m.config, config];
    IF Resident = NIL THEN Resident ← m ELSE lastResident.link ← m;
    lastResident ← m;
    END;

  AddSwappedIn: PROCEDURE [module, config: STRING] =
    BEGIN OPEN SystemDefs;
    m: ModuleHandle;
    m ← AllocateHeapNode[SIZE[ModuleObject]];
    m↑ ← [link: NIL, frame: ControlDefs.NullGlobalFrame,
      module: AllocateHeapString[module.length],
      config: AllocateHeapString[config.length]];
    StringDefs.AppendString[m.module, module];
    StringDefs.AppendString[m.config, config];
    IF SwappedIn = NIL THEN SwappedIn ← m ELSE lastSwappedIn.link ← m;
    lastSwappedIn ← m;
    END;

  AddNoTrap: PROCEDURE [module, config: STRING] =
    BEGIN OPEN SystemDefs;
    m: ModuleHandle;
    m ← AllocateHeapNode[SIZE[ModuleObject]];
    m↑ ← [link: NIL, frame: ControlDefs.NullGlobalFrame,
      module: AllocateHeapString[module.length],
      config: AllocateHeapString[config.length]];
    StringDefs.AppendString[m.module, module];
    StringDefs.AppendString[m.config, config];
    IF NoTrap = NIL THEN NoTrap ← m ELSE lastNoTrap.link ← m;
    lastNoTrap ← m;
    END;

  UserControl: PUBLIC PROCEDURE RETURNS [f: ControlDefs.GlobalFrameHandle] =
    BEGIN OPEN Control;
    IF Control = NIL THEN
      BEGIN
      IODefs.WriteLine["Warning: Null control module!"L];
      RETURN[ControlDefs.NullGlobalFrame];
      END;
    IF config.length # 0 THEN SetConfig[config];
    f ← frame ← Frame[module];
    IF frame = ControlDefs.NullGlobalFrame THEN
      IODefs.WriteLine["Warning: Null control module!"L];
    IF config.length # 0 THEN ResetConfig[];
    END;
```

```
NubFrame: PUBLIC PROCEDURE RETURNS [f: ControlDefs.GlobalFrameHandle] =
  BEGIN OPEN Nub;
  IF Nub = NIL THEN
    BEGIN
    IODefs.WriteLine["Warning: Null Nub!"L];
    RETURN[ControlDefs.NullGlobalFrame];
    END;
  IF config.length # 0 THEN SetConfig[config];
  f ← frame ← Frame[module];
  IF frame = ControlDefs.NullGlobalFrame THEN
    IODefs.WriteLine["Warning: Null Nub!"L];
  IF config.length # 0 THEN ResetConfig[];
  END;

WartFrame: PUBLIC PROCEDURE RETURNS [f: ControlDefs.GlobalFrameHandle] =
  BEGIN OPEN Wart;
  IF Wart = NIL THEN
    BootmesaDefs.BootmesaError["Error No Wart"L];
  IF config.length # 0 THEN SetConfig[config];
  f ← frame ← Frame[module];
  IF frame = ControlDefs.NullGlobalFrame THEN
    BootmesaModuleError["Error Can't find Wart: "L, config, module];
  IF config.length # 0 THEN ResetConfig[];
  END;

LookUpResidentModules: PUBLIC PROCEDURE =
  BEGIN
  m: ModuleHandle;
  FOR m ← Resident, m.link UNTIL m = NIL DO
    OPEN m;
    IF config.length # 0 THEN SetConfig[config];
    frame ← Frame[module];
    IF frame = ControlDefs.NullGlobalFrame THEN
      BootmesaModuleError["Can't find module: "L, config, module];
    IF config.length # 0 THEN ResetConfig[];
    ENDLOOP;
  END;

EnumerateResidentModules: PUBLIC PROCEDURE [
  proc: PROCEDURE [ControlDefs.GlobalFrameHandle] RETURNS [BOOLEAN]]
  RETURNS [ControlDefs.GlobalFrameHandle] =
  BEGIN
  m: ModuleHandle;
  FOR m ← Resident, m.link UNTIL m = NIL DO
    IF proc[m.frame] THEN RETURN[m.frame];
    ENDLOOP;
  RETURN[ControlDefs.NullGlobalFrame]
  END;

SwapInUserCode: PUBLIC PROCEDURE =
  BEGIN
  m: ModuleHandle;
  FOR m ← SwappedIn, m.link UNTIL m = NIL DO
    OPEN m;
    IF config.length # 0 THEN SetConfig[config];
    frame ← Frame[module];
    IF frame = ControlDefs.NullGlobalFrame THEN
      BootmesaModuleError["Can't find module: "L, config, module];
    SwapInFrame[frame];
    IF config.length # 0 THEN ResetConfig[];
    ENDLOOP;
  END;

NumberSwappedIn: PUBLIC PROCEDURE RETURNS [n: CARDINAL] =
  BEGIN
  m: ModuleHandle;
  n ← 0;
  FOR m ← SwappedIn, m.link UNTIL m = NIL DO n ← n + 1; ENDLOOP;
  RETURN[n]
  END;

TurnOffStartTrap: PUBLIC PROCEDURE =
  BEGIN
  m: ModuleHandle;
  codebase: RECORD [fill: [0..77777B), trap: BOOLEAN];
  codeseg: FakeSegDefs.FakeSegmentHandle;
```

```
    FOR m ← NoTrap, m.link UNTIL m = NIL DO
      OPEN m;
      IF config.length # 0 THEN SetConfig[config];
      frame ← Frame[module];
      IF frame = ControlDefs.NullGlobalFrame THEN
        BootmesaModuleError["Can't find module: "L, config, module];
      codebase ← BootCacheDefs.READ[@frame.code];
      codeseg ← BootCacheDefs.READ[@frame.codesegment];
      IF ~codeseg.SwappedIn THEN
        BootmesaModuleError["Module not swapped in: "L, config, module];
      codebase.trap ← FALSE;
      codebase ← LOOPHOLE[LOOPHOLE[codebase, CARDINAL]+ codeseg.VMaddress];
      BootCacheDefs.WRITE[@frame.code, codebase];
      IF config.length # 0 THEN ResetConfig[];
      ENDLOOP;
    END;

  UnlockUserCode: PUBLIC PROCEDURE =
    BEGIN
    m: ModuleHandle;
    FOR m ← SwappedIn, m.link UNTIL m = NIL DO
      UnlockFrame[m.frame];
      ENDLOOP;
    END;

  CloseSource: PUBLIC PROCEDURE =
    BEGIN OPEN SystemDefs;
    m: ModuleHandle;
    source.destroy[source];
    FOR m ← Resident, Resident UNTIL m = NIL DO
      Resident ← m.link;
      FreeHeapString[m.config];
      FreeHeapString[m.module];
      FreeHeapNode[m];
      ENDLOOP;
    FOR m ← SwappedIn, SwappedIn UNTIL m = NIL DO
      SwappedIn ← m.link;
      FreeHeapString[m.config];
      FreeHeapString[m.module];
      FreeHeapNode[m];
      ENDLOOP;
    FOR m ← NoTrap, NoTrap UNTIL m = NIL DO
      NoTrap ← m.link;
      FreeHeapString[m.config];
      FreeHeapString[m.module];
      FreeHeapNode[m];
      ENDLOOP;
    IF Nub # NIL THEN
      BEGIN
      FreeHeapString[Nub.config];
      FreeHeapString[Nub.module];
      FreeHeapNode[Nub];
      Nub ← NIL;
      END;
    IF Wart # NIL THEN
      BEGIN
      FreeHeapString[Wart.config];
      FreeHeapString[Wart.module];
      FreeHeapNode[Wart];
      Wart ← NIL;
      END;
    IF Control # NIL THEN
      BEGIN
      FreeHeapString[Control.config];
      FreeHeapString[Control.module];
      FreeHeapNode[Control];
      Control ← NIL;
      END;
    END;

  END...
```