DIRECTORY
IODefs: FROM "iodefs",
MiscDefs: FROM "miscdefs",
SegmentDefs: FROM "segmentdefs",
StreamDefs: FROM "streamdefs",
StringDefs: FROM "stringdefs",
SystemDefs: FROM "systemdefs",
TimeDefs: FROM "timedefs";

DEFINITIONS FROM StreamDefs, SegmentDefs;

OpDefsGenerator: PROGRAM
    IMPORTS IODefs, MiscDefs, StreamDefs, SegmentDefs, StringDefs, SystemDefs, TimeDefs =
BEGIN

CompStrDesc: TYPE = RECORD[
    offset, length: CARDINAL];

nChars: CARDINAL;
InStream, apOutStream, amOutStream, bOutStream: StreamHandle;

numopcodes: CARDINAL = 256;
opcode: TYPE = [0..numopcodes];

StringData: ARRAY opcode OF STRING;

Name: PROCEDURE [s: STRING] =
BEGIN
    c: CHARACTER;
    nc: CARDINAL + 0;
    CollectingChars: BOOLEAN + FALSE;
    s.length + 0;

    DO
        IF InStream.endof[InStream] THEN RETURN;
        c + InStream.get[InStream];
        SELECT c FROM
        IODefs.SP, IODefs.TAB, IODefs.CR =>
            IF CollectingChars THEN EXIT;
            IN [0..9] =>
                BEGIN
                    IF CollectingChars THEN EXIT;
                    SetIndex[InStream, ModifyIndex[GetIndex[InStream],-1]];
                    EXIT
                END;
                StringDefs.AppendChar[s,c];
            END;
            IN [A..Z], IN [a..z] =>
                BEGIN
                    CollectingChars+TRUE;
                    StringDefs.AppendChar[s,c];
                END;
        ENDCASE = SIGNAL SyntaxError;
    ENDLOOP;
    nChars + nChars + s.length;
RETURN
END;

Atom: PROCEDURE [s: STRING, del: CHARACTER] =
BEGIN
    c: CHARACTER;
    nc: CARDINAL + 0;
    CollectingChars: BOOLEAN + FALSE;

    DO
        IF InStream.endof[InStream] THEN SIGNAL SyntaxError;
        c + InStream.get[InStream];
        SELECT c FROM
        IODefs.SP, IODefs.CR =>
            IF CollectingChars THEN EXIT;
            IN [0..9], IN [A..Z], IN [a..z] =>
                BEGIN
                    s[nc] + c; nc + nc+1; CollectingChars+TRUE END;
ENDCASE => EXIT;
ENDLOOP;
s.length + nc:
IF c # del THEN SIGNAL SyntaxError;
RETURN
END;

SyntaxError: SIGNAL = CODE;

CollectOpData: PROCEDURE =
BEGIN OPEN SystemDefs;
i: opcode:
name: STRING [20];
s: STRING [8];
octal, decimal: CARDINAL;
Crcount: CARDINAL + 0;
push: POINTER TO ARRAY opcode OF CARDINAL + AllocateSegment[numopcodes];
pop: POINTER TO ARRAY opcode OF CARDINAL + AllocateSegment[numopcodes];
len: POINTER TO ARRAY opcode OF CARDINAL + AllocateSegment[numopcodes];
mark: POINTER TO ARRAY opcode OF CHARACTER + AllocateSegment[numopcodes];
	MiscDefs.Zero[push,numopcodes];
	MiscDefs.Zero[pop,numopcodes];
	MiscDefs.Zero[len,numopcodes];
	MiscDefs.SetBlock[mark,'F,numopcodes];

FOR i IN opcode DO
IF StringData[i] # NIL THEN
BEGIN FreeHeapString[StringData[i]]: StringData[i] + NIL END;
ENDLOOP;

nChars + 0:
UNTIL Crcount = 3 DO
IF InStream.get[InStream] = IODefs.CR THEN Crcount + Crcount+1;
ENDLOOP;

THROUGH opcode DO
Name[name]: IF InStream.endof[InStream] THEN EXIT;
Atom[s, ' ']:
octal + StringDefs.StringToNumber[s, 8];
Atom[s, ' ']:
decimal + StringDefs.StringToNumber[s, 10];
IF decimal = 0 THEN decimal + octal
ELSE IF octal = 0 THEN octal + decimal;
IF decimal # octal THEN SIGNAL OctalDecimalError[octal];
IF name.length # 0 THEN
BEGIN
StringData[octal] + SystemDefs.AllocateHeapString[name.length];
StringDefs.AppendString[StringData[octal],name];
END;

Atom[s, ' ']:
push[octal] + StringDefs.StringToOctal[s];
Atom[s, ' ']:
pop[octal] + StringDefs.StringToOctal[s];
Atom[s, ' ']:
len[octal] + StringDefs.StringToOctal[s];
Atom[s, ' ']:
mark[octal] + s[0];
ENDLOOP;

FOR i IN opcode DO
IF i MOD 4 = 0 THEN
BEGIN
f: IODefs.NumberFormat = [8,TRUE,TRUE,3];
OutString["--"L];
OutName[StringData[i],12];
OutName[StringData[i+1],13];
OutName[StringData[i+2],13];
OutName[StringData[i+3],13];
OutString[" "L]:
OutNumF[i,f]: OutChar["-"]: OutNumF[i+3,f]: OutString[""
"L]:
END;

OutString[" Q"L]:
OutNumF[i]: OutChar[","];
OutNumF[i]: OutChar[","];
OutNumF[i]: OutChar[","];
OutChar[mark[i]]: OutChar[" "];
IF i MOD 4 = 3 THEN
BEGIN
  IF i = LAST[opcode] THEN OutString["":"L]
  ELSE OutChar[','];
  OutChar[IODefs.CR];
  END
  ELSE OutChar[','];
  ENDLOOP;
SystemDefs.FreeSegment[push];
SystemDefs.FreeSegment[pop];
SystemDefs.FreeSegment[len];
RETURN
END;

OctalDecimalError: SIGNAL [CARDINAL] = CODE;
OpNameTooLong: ERROR [CARDINAL] = CODE;

OutStrings: PROCEDURE •
BEGIN
tSH: StreamHandle;
charpos: CARDINAL + 0;
i: opcode;
j: CARDINAL;

bOutStream.reset[bOutStream];
bOutStream.put[bOutStream, numopcodes*SIZE[CompStrDesc]+1];
FOR i IN opcode DO
  bOutStream.put[bOutStream, charpos];
  j - IF StringData[i] # NIL THEN StringData[i].length ELSE 0;
  bOutStream.put[bOutStream, j];
  charpos - charpos + j;
  ENDLOOP;
  bOutStream.put[bOutStream, nChars];
  bOutStream.put[bOutStream, nChars];
CleanupDiskStream[bOutStream];
tSH + CreateByteStream[OutFH, Write+Append];
SetIndex[tSH, GetIndex[bOutStream]];
bOutStream.reset[bOutStream];
bOutStream.destroy[bOutStream];
bOutStream + tSH;
FOR i IN opcode DO
  IF StringData[i] # NIL THEN
    BEGIN
      FOR j IN [0..StringData[i].length)
        DO bOutStream.put[bOutStream, StringData[i][j]]; ENDLOOP;
    END;
  ENDLOOP;
  bOutStream.destroy[bOutStream];
RETURN
END;

OutOpParams: PROCEDURE •
BEGIN
time: STRING + [20];
apOutStream.reset[apOutStream];
OutString["": -- generated by OpDefsGenerator "L];
OutString[time];
OutString["
Q: TYPE = PRIVATE RECORD [
  push: [0..3], pop: [0..7], length: [0..3], mark: BOOLEAN];
T: BOOLEAN = TRUE; F: BOOLEAN = FALSE;
OpParms: PRIVATE ARRAY [0..256] OF Q = [
"L];
CollectOpData[];
apOutStream.destroy[apOutStream];
RETURN
END;

OutMopcodes: PROCEDURE •
BEGIN
  i, j: CARDINAL;
i: opcode;
time: STRING + [20];
time.length + time.length = 3;
amOutStream.reset[amOutStream];
mOutString" -- generated by OpDefsGenerator "L;"
mOutString[time];
mOutChar[IODefs.CR];
mOutString[modulename];
mOutString["": DEFINITIONS =
BEGIN
op: TYPE = [0..400B];
"L];
FOR i IN opcode DO
  IF StringData[i] # NIL AND (1 + StringData[i].length) # 0 THEN
    BEGIN
      IF 1 > 10 THEN ERROR OpNameTooLong[i];
      FOR j IN (1..10) DO mOutChar[' '] ENDLOOP;
      mOutString[prefixString];
      mOutString[StringData[i]]; mOutChar[']';
      mOutString["": op = "L];
      IODefs.OutNumber[amOutStream, i, [8,FALSE,FALSE,3]]; mOutChar['B];
      ENDLOOP:
    ENDLOOP:
 ELSE FOR j IN [0..22) DO mOutChar[' '] ENDLOOP;
    IF (i MOD 4) # 3 THEN mOutChar['] ' ELSE mOutChar[IODefs.CR];
    ENDLOOP:
  mOutString["END..."
"L];
amOutStream.destroy[amOutStream];
RETURN
END:

OutName: PROCEDURE [s: STRING, n: CARDINAL] =
BEGIN
  1: CARDINAL = IF s = NIL THEN 0 ELSE s.length;
  THROUGH [1..n] DO OutChar[IODefs.SP]; ENDLOOP;
  OutString[s];
RETURN
END:

OutNum: PROCEDURE [n: CARDINAL] =
BEGIN
  IODefs.OutNumber[apOutStream, n, [10,FALSE,FALSE,1]]; RETURN
END:

OutNumF: PROCEDURE [n: CARDINAL, f: IODefs.NumberFormat] =
BEGIN
  IODefs.OutNumber[apOutStream, n, f]; RETURN
END:

OutString: PROCEDURE [s: STRING] =
BEGIN
  i: CARDINAL;
  IF s # NIL THEN
    FOR i IN [0..s.length) DO apOutStream.put[apOutStream, s[i]];ENDLOOP;
  RETURN
END:

mOutString: PROCEDURE [s: STRING] =
BEGIN
  i: CARDINAL;
  IF s # NIL THEN
    FOR i IN [0..s.length) DO amOutStream.put[amOutStream, s[i]];ENDLOOP;
  RETURN
END:

OutChar: PROCEDURE [c: CHARACTER] =
BEGIN apOutStream.put[apOutStream, c]; RETURN END:
mOutChar: PROCEDURE [c: CHARACTER] =
BEGIN amOutStream.put[amOutStream, c]; RETURN END;

DefaultNames: TYPE = {infile, apoutfile, amoutfile, boutfile, modulename, prefix};

MopDefaults: ARRAY DefaultNames OF STRING = [

FopDefaults: ARRAY DefaultNames OF STRING = [
"FOpCodes.txt", "FOpParams", "FOpCodes.mesa", "FOpNames.binary", "FOpCodes", "q"];

infile: STRING + [40];
apoutfile: STRING + [40];
amoutfile: STRING + [40];
boutfile: STRING + [40];
modulename: STRING + [40];
prefixString: STRING + [10];

outFH: FileHandle;

SetDefaults: PROCEDURE [p: POINTER TO ARRAY DefaultNames OF STRING] =
BEGIN OPEN StringDefs;
infile.length=0; AppendString[infile, p[infile]];
apoutfile.length=0; AppendString[apoutfile, p[apoutfile]];
amoutfile.length=0; AppendString[amoutfile, p[amoutfile]];
boutfile.length=0; AppendString[boutfile, p[boutfile]];
modulename.length=0; AppendString[modulename, p[modulename]];
prefixString.length=0; AppendString[prefixString, p[prefix]];
END;

GetResponse: PROCEDURE[prompt, response: STRING] =
BEGIN OPEN IODefs;
WriteString[prompt];
ReadID[response];
WriteChar[CR];
END;

MiscDefs.SetBlock[BASE[StringData], NIL, numopcodes];

IODefs.WriteString["Mesa OpData Generator "];
DO
DO OPEN IODefs;
WriteString[
Mopdata, Fopdata, or Quit: "];
SELECT ReadChar[] FROM
'M, M => BEGIN SetDefaults[@MopDefaults]; EXIT END;
'F, F => BEGIN SetDefaults[@FopDefaults]; EXIT END;
'q, Q => GOTO done;
ENDCASE;
ENDLOOP;
IODefs.WriteChar[IODefs.CR];
IODefs.WriteLine["Use escape key to get defaults"];
GetResponse["Input file: ", infile];
IF infile.length = 0 THEN EXIT;
GetResponse[" OpParams file: ", apoutfile];
GetResponse[" Mopcodes file: ", amoutfile];
GetResponse[" Module name (capitalize correctly): ", modulename];
GetResponse[" Prefix with: ", prefixString];
GetResponse[" binary file for OpName strings: ", boutfile];
bOutStream + CreateWordStream[outFH + NewFile[boutfile, Write+Append, DefaultAccess], Write+Append];
amOutStream + NewByteStream[amoutfile, Write+Append];
apOutStream + NewByteStream[apoutfile, Write+Append];
OutOpParams[]; OutStrings[]; OutMopcodes[];
InStream.destroy[{InStream}];
REPEAT done => NULL;
ENDLOOP;
END...