```
MODULE DevCPM;
(**

	project	= "BlackBox"
	organization	= "www.oberon.ch"
	contributors	= "Oberon microsystems"
	version	= "System/Rsrc/About"
	copyright	= "System/Rsrc/About"
	license	= "Docu/BB-License"
	references	= "http://e-collection.library.ethz.ch/eserv/eth:39386/eth-39386-02.pdf"
	changes	= " "
	issues	= " "

**)

	IMPORT SYSTEM, Kernel, Files, Stores, Models, TextModels, StdLog, DevMarkers, Dialog;

	CONST
		ProcSize* = 4;	(* PROCEDURE type *)
		PointerSize* = 4;	(* POINTER type *)
		DArrSizeA* = 8;	(* dyn array descriptor *)
		DArrSizeB* = 4;	(* size = A + B * typ.n *)

		MaxSet* = 31;
		MaxIndex* = 7FFFFFFFH;	(* maximal index value for array declaration *)

		MinReal32Pat = 0FF7FFFFFH;	(* most positive, 32-bit pattern *)
		MinReal64PatL = 0FFFFFFFFH;	(* most  negative, lower 32-bit pattern *)
		MinReal64PatH = 0FFEFFFFFH;	(* most  negative, higher 32-bit pattern *)
		MaxReal32Pat = 07F7FFFFFH;	(* most positive, 32-bit pattern *)
		MaxReal64PatL = 0FFFFFFFFH;	(* most positive, lower 32-bit pattern *)
		MaxReal64PatH = 07FEFFFFFH;	(* most positive, higher 32-bit pattern *)
		InfRealPat = 07F800000H;	(* real infinity pattern *)


		(* inclusive range of parameter of standard procedure HALT *)
		MinHaltNr* = 0;
		MaxHaltNr* = 128;

		(* inclusive range of register number of procedures SYSTEM.GETREG and SYSTEM.PUTREG *)
		MinRegNr* = 0;
		MaxRegNr* = 31;

		(* maximal value of flag used to mark interface structures *)
		MaxSysFlag* = 127;	(* shortint *)
		CProcFlag* = 1;	(* code procedures *)

		(* maximal condition value of parameter of SYSTEM.CC *)
		MaxCC* = 15;

		(* initialization of constant address, must be different from any valid constant address *)
		ConstNotAlloc* = -1;

		(* whether hidden pointer fields have to be nevertheless exported *)
		ExpHdPtrFld* = TRUE;
		HdPtrName* = "@ptr";

		(* whether hidden untagged pointer fields have to be nevertheless exported *)
		ExpHdUtPtrFld* = TRUE;
		HdUtPtrName* = "@utptr";
```

```
    (* whether hidden procedure fields have to be nevertheless exported (may be used for System.Free) *)
    ExpHdProcFld* = TRUE;
    HdProcName* = "@proc";

    (* whether hidden bound procedures have to be nevertheless exported *)
    ExpHdTProc* = FALSE;
    HdTProcName* = "@tproc";

    (* maximal number of exported stuctures: *)
    MaxStruct* = 16000;    (* must be < MAX(INTEGER) DIV 2 in object model *)

    (* maximal number of record extensions: *)
    MaxExts* = 15;    (* defined by type descriptor layout *)

    (* whether field leaf of pointer variable p has to be set to FALSE, when NEW(p) or SYSTEM.NEW(p, n) is used *)
    NEWusingAdr* = FALSE;

    (* special character (< " ") returned by procedure Get, if end of text reached *)
    Eot* = 0X;

    (* warnings *)
    longreal* = 0; largeint* = 1; realConst* = 2; copy* = 3; lchr* = 4; lentier* = 5; invar* = 6; outvar* = 7;

    (* language options *)
    interface* = 1;
    com* = 2; comAware* = 3;
    som* = 4; somAware* = 5; the language options are platform-specific
    oberon* = 6;
    java* = 7; javaAware* = 8;
    noCode* = 9;
    allSysVal* = 14;
    sysImp* = 15;
    trap* = 31;
    sys386 = 10; sys68k = 20;    (* processor type in options if system imported *)
                                   the language options are platform-specific
  CONST
    SFdir = "Sym";
    OFdir = "Code";
    SYSdir = "System";
    SFtag = 6F4F5346H;    (* symbol file tag *)
    OFtag = 6F4F4346H;    (* object file tag *)
    maxErrors = 64;

  VAR
    LEHost*: BOOLEAN;    (* little or big endian host *)
    MinReal32*, MaxReal32*, InfReal*,
    MinReal64*, MaxReal64*: REAL;
    noerr*: BOOLEAN;    (* no error found until now *)
    curpos*, startpos*, errpos*: INTEGER;    (* character, start, and error position in source file *)
    searchpos*: INTEGER;    (* search position in source file *)
    errors*: INTEGER;
    breakpc*: INTEGER;    (* set by OPV.Init *)
    options*: SET;    (* language options *)  the language options are platform-specific
    file*: Files.File;    (* used for sym file import *)
    codeDir*: ARRAY 16 OF CHAR;
    symDir*: ARRAY 16 OF CHAR;
    checksum*: INTEGER;    (* symbol file checksum *)

    lastpos: INTEGER;
    ObjFName: Files.Name;
```

```
        in: TextModels.Reader;
        oldSymFile, symFile, objFile: Files.File;
        inSym: Files.Reader;
        outSym, outObj: Files.Writer;

        errNo, errPos: ARRAY maxErrors OF INTEGER;

        lineReader: TextModels.Reader;
        lineNum: INTEGER;

        crc32tab: ARRAY 256 OF INTEGER;


    PROCEDURE^ err* (n: INTEGER);

    PROCEDURE Init* (source: TextModels.Reader; logtext: TextModels.Model);
    BEGIN
        in := source;
        DevMarkers.Unmark(in.Base());
        noerr := TRUE; options := {};
        curpos := in.Pos(); errpos := curpos; lastpos := curpos - 11; errors := 0;
        codeDir := OFdir; symDir := SFdir
    END Init;

    PROCEDURE Close*;
    BEGIN
        oldSymFile := NIL; inSym := NIL;
        symFile := NIL; outSym := NIL;
        objFile := NIL; outObj := NIL;
        in := NIL; lineReader := NIL
    END Close;

    PROCEDURE Get* (VAR ch: CHAR);
    BEGIN
        REPEAT in.ReadChar(ch); INC(curpos) UNTIL ch # TextModels.viewcode;
    END Get;

    PROCEDURE LineOf* (pos: INTEGER): INTEGER;
        VAR ch: CHAR;
    BEGIN
        IF lineReader = NIL THEN lineReader := in.Base().NewReader(NIL); lineReader.SetPos(0); lineNum := 0 END;
        IF lineReader.Pos() > pos THEN lineReader.SetPos(0); lineNum := 0 END;
        WHILE lineReader.Pos() < pos DO
            lineReader.ReadChar(ch);
            IF ch = 0DX THEN INC(lineNum) END
        END;
        RETURN lineNum
    END LineOf;

    PROCEDURE LoWord (r: REAL): INTEGER;
        VAR x: INTEGER;
    BEGIN
        x := SYSTEM.ADR(r);
        IF ~LEHost THEN INC(x, 4) END;
        SYSTEM.GET(x, x);
        RETURN x
    END LoWord;

    PROCEDURE HiWord (r: REAL): INTEGER;
        VAR x: INTEGER;
```

```
BEGIN
    x := SYSTEM.ADR(r);
    IF LEHost THEN INC(x, 4) END;
    SYSTEM.GET(x, x);
    RETURN x
END HiWord;

PROCEDURE Compound (lo, hi: INTEGER): REAL;
    VAR r: REAL;
BEGIN
    IF LEHost THEN
        SYSTEM.PUT(SYSTEM.ADR(r), lo); SYSTEM.PUT(SYSTEM.ADR(r) + 4, hi)
    ELSE
        SYSTEM.PUT(SYSTEM.ADR(r) + 4, lo); SYSTEM.PUT(SYSTEM.ADR(r), hi)
    END;
    RETURN r
END Compound;


(* sysflag control *)  Sysflag procedures can be part of parser, platform-specific part
These procedures can be place out and the parser can have the procedural hooks for it
PROCEDURE ValidGuid* (IN str: ARRAY OF SHORTCHAR): BOOLEAN;
    VAR i: INTEGER; ch: SHORTCHAR;
BEGIN
    IF (LEN(str$) # 38) OR (str[0] # "{") & (str[37] # "}") THEN RETURN FALSE END;
    i := 1;
    WHILE i < 37 DO
        ch := str[i];
        IF (i = 9) OR (i = 14) OR (i = 19) OR (i = 24) THEN
            IF ch # "-" THEN RETURN FALSE END
        ELSE
            IF (ch < "0") OR (ch > "9") & (CAP(ch) < "A") OR (CAP(ch) > "Z") THEN RETURN FALSE END
        END;
        INC(i)
    END;
    RETURN TRUE
END ValidGuid;

PROCEDURE GetProcSysFlag* (IN id: ARRAY OF SHORTCHAR; num: SHORTINT; VAR flag: SHORTINT);
BEGIN
    IF id # "" THEN
        IF id = "code" THEN num := 1
        ELSIF id = "callback" THEN num := 2
        ELSIF id = "nostkchk" THEN num := 4
        ELSIF id = "ccall" THEN num := -10
        ELSIF id = "guarded" THEN num := 8
        ELSIF id = "noframe" THEN num := 16
        ELSIF id = "native" THEN num := -33
        ELSIF id = "bytecode" THEN num := -35
        END
    END;
    IF (options * {sysImp, sys386, sys68k} # {}) & ((num = 1) OR (num = 2)) THEN INC(flag, num)
    ELSIF (sys68k IN options) & (num = 4) THEN INC(flag, num)
    ELSIF (options * {sys386, interface} # {}) & (num = -10) & (flag = 0) THEN flag := -10
    ELSIF (options * {sys386, com} # {}) & (num = 8) & (flag = 0) THEN flag := 8
    ELSIF (options * {sysImp, sys386} # {}) & (num = 16) & (flag = 0) THEN flag := 16
    ELSIF ({sysImp, java} - options = {}) & ((num= -33) OR (num = -35)) & (flag = 0) THEN flag := num
    ELSE err(225); flag := 0
    END
END GetProcSysFlag;
```

```
PROCEDURE GetVarParSysFlag* (IN id: ARRAY OF SHORTCHAR; num: SHORTINT; VAR flag: SHORTINT);
    VAR old: SHORTINT;
BEGIN
    old := flag; flag := 0;
    IF (options * {sys386, sys68k, interface, com} # {}) THEN
        IF (num = 1) OR (id = "nil") THEN
            IF ~ODD(old) THEN flag := SHORT(old + 1) END
        ELSIF ((num = 2) OR (id = "in")) & (oberon IN options) THEN
            IF old <= 1 THEN flag := SHORT(old + 2) END
        ELSIF ((num = 4) OR (id = "out")) & (oberon IN options) THEN
            IF old <= 1 THEN flag := SHORT(old + 4) END
        ELSIF ((num = 8) OR (id = "new")) & (options * {com, interface} # {}) THEN
            IF old <= 1 THEN flag := SHORT(old + 8) END
        ELSIF ((num = 16) OR (id = "iid")) & (com IN options) THEN
            IF old <= 1 THEN flag := SHORT(old + 16) END
        END
    END;
    IF flag = 0 THEN err(225) END
END GetVarParSysFlag;

PROCEDURE GetRecordSysFlag* (IN id: ARRAY OF SHORTCHAR; num: SHORTINT; VAR flag: SHORTINT);
    VAR old: SHORTINT;
BEGIN
    old := flag; flag := 0;
    IF (num = 1) OR (id = "untagged") THEN
        IF (options * {sysImp, sys386, sys68k, interface, com, som} # {}) & (old = 0) THEN flag := 1 END
    ELSIF (num = 3) OR (id = "noalign") THEN
        IF (options * {sys386, sys68k, interface, com, som} # {}) & (old = 0) THEN flag := 3 END
    ELSIF (num = 4) OR (id = "align2") THEN
        IF (options * {sys386, interface, com} # {}) & (old = 0) THEN flag := 4 END
    ELSIF (num = 5) OR (id = "align4") THEN
        IF (options * {sys386, interface, com} # {}) & (old = 0) THEN flag := 5 END
    ELSIF (num = 6) OR (id = "align8") THEN
        IF (options * {sys386, interface, com} # {}) & (old = 0) THEN flag := 6 END
    ELSIF (num = 7) OR (id = "union") THEN
        IF (options * {sys386, sys68k, interface, com} # {}) & (old = 0) THEN flag := 7 END
    ELSIF (num = 10) OR (id = "interface") OR ValidGuid(id) THEN
        IF (com IN options) & (old = 0) THEN flag := 10 END
    ELSIF (num = -11) OR (id = "jint") THEN
        IF (java IN options) & (old = 0) THEN flag := -11 END
    ELSIF (num = -13) OR (id = "jstr") THEN
        IF (java IN options) & (old = 0) THEN flag := -13 END
    ELSIF (num = 20) OR (id = "som") THEN
        IF (som IN options) & (old = 0) THEN flag := 20 END
    END;
    IF flag = 0 THEN err(225) END
END GetRecordSysFlag;

PROCEDURE GetArraySysFlag* (IN id: ARRAY OF SHORTCHAR; num: SHORTINT; VAR flag: SHORTINT);
    VAR old: SHORTINT;
BEGIN
    old := flag; flag := 0;
    IF (num = 1) OR (id = "untagged") THEN
        IF (options * {sysImp, sys386, sys68k, interface, com, som} # {}) & (old = 0) THEN flag := 1 END
    ELSIF (num = -12) OR (id = "jarr") THEN
        IF (java IN options) & (old = 0) THEN flag := -12 END
    ELSIF (num = -13) OR (id = "jstr") THEN
        IF (java IN options) & (old = 0) THEN flag := -13 END
    END;
```

```
	IF flag = 0 THEN err(225) END
END GetArraySysFlag;

PROCEDURE GetPointerSysFlag* (IN id: ARRAY OF SHORTCHAR; num: SHORTINT; VAR flag: SHORTINT);
	VAR old: SHORTINT;
BEGIN
	old := flag; flag := 0;
	IF (num = 1) OR (id = "untagged") THEN
		IF (options * {sys386, sys68k, interface, com, som} # {}) & (old = 0) THEN flag := 1 END
	ELSIF (num = 2) OR (id = "handle") THEN
		IF (sys68k IN options) & (old = 0) THEN flag := 2 END
	ELSIF (num = 10) OR (id = "interface") THEN
		IF (com IN options) & (old = 0) THEN flag := 10 END
	ELSIF (num = 20) OR (id = "som") THEN
		IF (som IN options) & (old = 0) THEN flag := 20 END
	END;
	IF flag = 0 THEN err(225) END
END GetPointerSysFlag;

PROCEDURE GetProcTypSysFlag* (IN id: ARRAY OF SHORTCHAR; num: SHORTINT; VAR flag: SHORTINT);
BEGIN
	IF ((num = -10) OR (id = "ccall")) & (options * {sys386, interface} # {}) THEN flag := -10
	ELSE err(225); flag := 0
	END
END GetProcTypSysFlag;

PROCEDURE PropagateRecordSysFlag* (baseFlag: SHORTINT; VAR flag: SHORTINT);
BEGIN
	IF (baseFlag = 1) OR (baseFlag >= 3) & (baseFlag <= 7) THEN	(* propagate untagged .. union *)
		IF flag = 0 THEN flag := baseFlag
		ELSIF (flag = 6) & (baseFlag < 6) THEN (* OK *)	(* special case for 8 byte aligned records *)
		ELSIF flag # baseFlag THEN err(225); flag := 0
		END
	ELSIF (baseFlag # 10) & (flag = 10) THEN err(225)
	END
END PropagateRecordSysFlag;

PROCEDURE PropagateRecPtrSysFlag* (baseFlag: SHORTINT; VAR flag: SHORTINT);
BEGIN
	IF (baseFlag = 1) OR (baseFlag >= 3) & (baseFlag <= 7) THEN	(* pointer to untagged .. union is untagged *)
		IF flag = 0 THEN flag := 1
		ELSIF (flag # 1) & (flag # 2) THEN err(225); flag := 0
		END
	ELSIF baseFlag = 10 THEN	(* pointer to interface is interface *)
		IF flag = 0 THEN flag := 10
		ELSIF flag # 10 THEN err(225); flag := 0
		END
	ELSIF baseFlag = -11 THEN	(* pointer to java interface is java interface *)
		IF flag # 0 THEN err(225) END;
		flag := -11
	ELSIF baseFlag = -13 THEN	(* pointer to java string is java string *)
		IF flag # 0 THEN err(225) END;
		flag := -13
	END
END PropagateRecPtrSysFlag;

PROCEDURE PropagateArrPtrSysFlag* (baseFlag: SHORTINT; VAR flag: SHORTINT);
BEGIN
	IF baseFlag = 1 THEN	(* pointer to untagged or guid is untagged *)
		IF flag = 0 THEN flag := 1
```

```
        ELSIF (flag # 1) & (flag # 2) THEN err(225); flag := 0
        END
    ELSIF baseFlag = -12 THEN    (* pointer to java array is java array *)
        IF flag # 0 THEN err(225) END;
        flag := -12
    ELSIF baseFlag = -13 THEN    (* pointer to java string is java string *)
        IF flag # 0 THEN err(225) END;
        flag := -13
    END
END PropagateArrPtrSysFlag;


(* utf8 strings *)

PROCEDURE PutUtf8* (VAR str: ARRAY OF SHORTCHAR; val: INTEGER; VAR idx: INTEGER);
BEGIN
    ASSERT((val >= 0) & (val < 65536));
    IF val < 128 THEN
        str[idx] := SHORT(CHR(val)); INC(idx)
    ELSIF val < 2048 THEN
        str[idx] := SHORT(CHR(val DIV 64 + 192)); INC(idx);
        str[idx] := SHORT(CHR(val MOD 64 + 128)); INC(idx)
    ELSE
        str[idx] := SHORT(CHR(val DIV 4096 + 224)); INC(idx);
        str[idx] := SHORT(CHR(val DIV 64 MOD 64 + 128)); INC(idx);
        str[idx] := SHORT(CHR(val MOD 64 + 128)); INC(idx)
    END
END PutUtf8;

PROCEDURE GetUtf8* (IN str: ARRAY OF SHORTCHAR; VAR val, idx: INTEGER);
    VAR ch: SHORTCHAR;
BEGIN
    ch := str[idx]; INC(idx);
    IF ch < 80X THEN
        val := ORD(ch)
    ELSIF ch < 0E0X THEN
        val := ORD(ch) - 192;
        ch := str[idx]; INC(idx); val := val * 64 + ORD(ch) - 128
    ELSE
        val := ORD(ch) - 224;
        ch := str[idx]; INC(idx); val := val * 64 + ORD(ch) - 128;
        ch := str[idx]; INC(idx); val := val * 64 + ORD(ch) - 128
    END
END GetUtf8;


(* log output *)

PROCEDURE LogW* (ch: CHAR);
BEGIN
    StdLog.Char(ch)
END LogW;

PROCEDURE LogWStr* (IN s: ARRAY OF CHAR);
BEGIN
    StdLog.String(s)
END LogWStr;

PROCEDURE LogWPar* (IN key: ARRAY OF CHAR; IN p0, p1: ARRAY OF SHORTCHAR);
    VAR s0, s1, s: Dialog.String; res: INTEGER;
```

```
	BEGIN
		Kernel.Utf8ToString(p0, s0, res);
		Kernel.Utf8ToString(p1, s1, res);
		Dialog.MapParamString(key, s0, s1, "", s);
		StdLog.String(s)
	END LogWPar;

	PROCEDURE LogWNum* (i, len: INTEGER);
	BEGIN
		StdLog.Int(i)
	END LogWNum;

	PROCEDURE LogWLn*;
	BEGIN
		StdLog.Ln
	END LogWLn;
(*
	PROCEDURE LogW* (ch: CHAR);
	BEGIN
		out.WriteChar(ch);
	END LogW;

	PROCEDURE LogWStr* (s: ARRAY OF CHAR);
	BEGIN
		out.WriteString(s);
	END LogWStr;

	PROCEDURE LogWNum* (i, len: LONGINT);
	BEGIN
		out.WriteChar(" "); out.WriteInt(i);
	END LogWNum;

	PROCEDURE LogWLn*;
	BEGIN
		out.WriteLn;
		Views.RestoreDomain(logbuf.Domain())
	END LogWLn;
*)
	PROCEDURE Mark* (n, pos: INTEGER);
	BEGIN
		IF (n >= 0) & ~((oberon IN options) & (n >= 181) & (n <= 190)) THEN
			noerr := FALSE;
			IF pos < 0 THEN pos := 0 END;
			IF (pos < lastpos) OR (lastpos + 9 < pos) THEN
				lastpos := pos;
				IF errors < maxErrors THEN
					errNo[errors] := n; errPos[errors] := pos
				END;
				INC(errors)
			END;
			IF trap IN options THEN HALT(100) END;
		ELSIF (n <= -700) & (errors < maxErrors) THEN
			errNo[errors] := -n; errPos[errors] := pos; INC(errors)
		END
	END Mark;

	PROCEDURE err* (n: INTEGER);
	BEGIN
		Mark(n, errpos)
	END err;
	PROCEDURE InsertMarks* (text: TextModels.Model);
		VAR i, j, x, y, n: INTEGER; script: Stores.Operation;
	BEGIN
		n := errors;
		IF n > maxErrors THEN n := maxErrors END;
		(* sort *)
		i := 1;
		WHILE i < n DO
			x := errPos[i]; y := errNo[i]; j := i-1;
			WHILE (j >= 0) & (errPos[j] > x) DO errPos[j+1] := errPos[j]; errNo[j+1] := errNo[j]; DEC(j) END;
			errPos[j+1] := x; errNo[j+1] := y; INC(i)
		END;
		(* insert *)
		Models.BeginModification(Models.clean, text);
		Models.BeginScript(text, "#Dev:InsertMarkers", script);
		WHILE n > 0 DO DEC(n);
			DevMarkers.Insert(text, errPos[n], DevMarkers.dir.New(errNo[n]))
		END;
		Models.EndScript(text, script);
		Models.EndModification(Models.clean, text);
	END InsertMarks;


	(* fingerprinting *)

	PROCEDURE InitCrcTab;
		(* CRC32, high bit first, pre & post inverted *)
		CONST poly = {0, 1, 2, 4, 5, 7, 8, 10, 11, 12, 16, 22, 23, 26};	(* CRC32 polynom *)
		VAR x, c, i: INTEGER;
	BEGIN
		x := 0;
		WHILE x < 256 DO
			c := x * 1000000H; i := 0;
			WHILE i < 8 DO
				IF c < 0 THEN c := ORD(BITS(c * 2) / poly)
				ELSE c := c * 2
				END;
				INC(i)
			END;
			crc32tab[ORD(BITS(x) / BITS(255))] := ORD(BITS(c) / BITS(255));
			INC(x)
		END
	END InitCrcTab;

	PROCEDURE FPrint* (VAR fp: INTEGER; val: INTEGER);
		VAR c: INTEGER;
	BEGIN
(*
		fp := SYSTEM.ROT(ORD(BITS(fp) / BITS(val)), 1)	(* bad collision detection *)
*)
		(* CRC32, high bit first, pre & post inverted *)
		c := ORD(BITS(fp * 256) / BITS(crc32tab[ORD(BITS(fp DIV 1000000H) / BITS(val DIV 1000000H)) MOD 256]));
		c := ORD(BITS(c * 256) / BITS(crc32tab[ORD(BITS(c DIV 1000000H) / BITS(val DIV 10000H)) MOD 256]));
		c := ORD(BITS(c * 256) / BITS(crc32tab[ORD(BITS(c DIV 1000000H) / BITS(val DIV 100H)) MOD 256]));
		fp := ORD(BITS(c * 256) / BITS(crc32tab[ORD(BITS(c DIV 1000000H) / BITS(val)) MOD 256]));
	END FPrint;

	PROCEDURE FPrintSet* (VAR fp: INTEGER; set: SET);
	BEGIN FPrint(fp, ORD(set))
	END FPrintSet;

	PROCEDURE FPrintReal* (VAR fp: INTEGER; real: SHORTREAL);
	BEGIN FPrint(fp, SYSTEM.VAL(INTEGER, real))
	END FPrintReal;

	PROCEDURE FPrintLReal* (VAR fp: INTEGER; lr: REAL);
	BEGIN
		FPrint(fp, LoWord(lr)); FPrint(fp, HiWord(lr))
	END FPrintLReal;

	PROCEDURE ChkSum (VAR fp: INTEGER; val: INTEGER);	(* symbolfile checksum *)
	BEGIN
		(* same as FPrint, 8 bit only *)
		fp := ORD(BITS(fp * 256) / BITS(crc32tab[ORD(BITS(fp DIV 1000000H) / BITS(val)) MOD 256]))
	END ChkSum;



	(* compact format *)

	PROCEDURE WriteLInt (w: Files.Writer; i: INTEGER);
	BEGIN
		ChkSum(checksum, i);
		w.WriteByte(SHORT(SHORT(i MOD 256))); i := i DIV 256;
		ChkSum(checksum, i);
		w.WriteByte(SHORT(SHORT(i MOD 256))); i := i DIV 256;
		ChkSum(checksum, i);
		w.WriteByte(SHORT(SHORT(i MOD 256))); i := i DIV 256;
		ChkSum(checksum, i);
		w.WriteByte(SHORT(SHORT(i MOD 256)))
	END WriteLInt;

	PROCEDURE ReadLInt (r: Files.Reader; VAR i: INTEGER);
		VAR b: BYTE; x: INTEGER;
	BEGIN
		r.ReadByte(b); x := b MOD 256;
		ChkSum(checksum, b);
		r.ReadByte(b); x := x + 100H * (b MOD 256);
		ChkSum(checksum, b);
		r.ReadByte(b); x := x + 10000H * (b MOD 256);
		ChkSum(checksum, b);
		r.ReadByte(b); i := x + 1000000H * b;
		ChkSum(checksum, b)
	END ReadLInt;

	PROCEDURE WriteNum (w: Files.Writer; i: INTEGER);
	BEGIN	(* old format of Oberon *)
		WHILE (i < -64) OR (i > 63) DO ChkSum(checksum, i MOD 128 - 128); w.WriteByte(SHORT(SHORT(i MOD 128 -
128))); i := i DIV 128 END;
		ChkSum(checksum, i MOD 128);
		w.WriteByte(SHORT(SHORT(i MOD 128)))
	END WriteNum;

	PROCEDURE ReadNum (r: Files.Reader; VAR i: INTEGER);
		VAR b: BYTE; s, y: INTEGER;
	BEGIN
		s := 0; y := 0; r.ReadByte(b);
		IF ~r.eof THEN ChkSum(checksum, b) END;
		WHILE b < 0 DO INC(y, ASH(b + 128, s)); INC(s, 7); r.ReadByte(b); ChkSum(checksum, b) END;
		i := ASH((b + 64) MOD 128 - 64, s) + y;
	END ReadNum;

	PROCEDURE WriteNumSet (w: Files.Writer; x: SET);
	BEGIN
		WriteNum(w, ORD(x))
	END WriteNumSet;

	PROCEDURE ReadNumSet (r: Files.Reader; VAR x: SET);
		VAR i: INTEGER;
	BEGIN
		ReadNum(r, i); x := BITS(i)
	END ReadNumSet;

	PROCEDURE WriteReal (w: Files.Writer; x: SHORTREAL);
	BEGIN
		WriteLInt(w, SYSTEM.VAL(INTEGER, x))
	END WriteReal;

	PROCEDURE ReadReal (r: Files.Reader; VAR x: SHORTREAL);
		VAR i: INTEGER;
	BEGIN
		ReadLInt(r, i); x := SYSTEM.VAL(SHORTREAL, i)
	END ReadReal;

	PROCEDURE WriteLReal (w: Files.Writer; x: REAL);
	BEGIN
		WriteLInt(w, LoWord(x)); WriteLInt(w, HiWord(x))
	END WriteLReal;

	PROCEDURE ReadLReal (r: Files.Reader; VAR x: REAL);
		VAR h, l: INTEGER;
	BEGIN
		ReadLInt(r, l); ReadLInt(r, h); x := Compound(l, h)
	END ReadLReal;


	(* read symbol file *)

	PROCEDURE SymRCh* (VAR ch: SHORTCHAR);
		VAR b: BYTE;
	BEGIN
		inSym.ReadByte(b); ch := SHORT(CHR(b));
		ChkSum(checksum, b)
	END SymRCh;

	PROCEDURE SymRInt* (): INTEGER;
		VAR k: INTEGER;
	BEGIN
		ReadNum(inSym, k); RETURN k
	END SymRInt;

	PROCEDURE SymRSet* (VAR s: SET);
	BEGIN
		ReadNumSet(inSym, s)
	END SymRSet;

	PROCEDURE SymRReal* (VAR r: SHORTREAL);
	BEGIN
		ReadReal(inSym, r)
```

```
END SymRReal;

PROCEDURE SymRLReal* (VAR lr: REAL);
BEGIN
    ReadLReal(inSym, lr)
END SymRLReal;

PROCEDURE eofSF* (): BOOLEAN;
BEGIN
    RETURN inSym.eof
END eofSF;

PROCEDURE OldSym* (IN modName: ARRAY OF SHORTCHAR; VAR done: BOOLEAN);
    VAR tag, res: INTEGER; loc: Files.Locator; dir, name: Files.Name;
BEGIN
    done := FALSE;
    IF modName = "@file" THEN
        oldSymFile := file
    ELSE
        Kernel.Utf8ToString(modName, name, res); Kernel.SplitName(name, dir, name);
        Kernel.MakeFileName(name, Kernel.symType);
        loc := Files.dir.This(dir); loc := loc.This(symDir);
        oldSymFile := Files.dir.Old(loc, name, Files.shared);
        IF (oldSymFile = NIL) & (dir = "") THEN
            loc := Files.dir.This(SYSdir); loc := loc.This(symDir);
            oldSymFile := Files.dir.Old(loc, name, Files.shared)
        END
    END;
    IF oldSymFile # NIL THEN
        inSym := oldSymFile.NewReader(inSym);
        IF inSym # NIL THEN
            ReadLInt(inSym, tag);
            IF tag = SFtag THEN done := TRUE ELSE err(151) END
        END
    END
END OldSym;

PROCEDURE CloseOldSym*;
BEGIN
    IF oldSymFile # NIL THEN oldSymFile.Close; oldSymFile := NIL END
END CloseOldSym;


(* write symbol file *)

PROCEDURE SymWCh* (ch: SHORTCHAR);
BEGIN
    ChkSum(checksum, ORD(ch));
    outSym.WriteByte(SHORT(ORD(ch)))
END SymWCh;

PROCEDURE SymWInt* (i: INTEGER);
BEGIN
    WriteNum(outSym, i)
END SymWInt;

PROCEDURE SymWSet* (s: SET);
BEGIN
    WriteNumSet(outSym, s)
END SymWSet;
```

```
PROCEDURE SymWReal* (r: SHORTREAL);
BEGIN
    WriteReal(outSym, r)
END SymWReal;

PROCEDURE SymWLReal* (r: REAL);
BEGIN
    WriteLReal(outSym, r)
END SymWLReal;

PROCEDURE SymReset*;
BEGIN
    outSym.SetPos(4)
END SymReset;

PROCEDURE NewSym* (IN modName: ARRAY OF SHORTCHAR);
    VAR res: INTEGER; loc: Files.Locator; dir: Files.Name;
BEGIN
    Kernel.Utf8ToString(modName, ObjFName, res); Kernel.SplitName(ObjFName, dir, ObjFName);
    loc := Files.dir.This(dir); loc := loc.This(symDir);
    symFile := Files.dir.New(loc, Files.ask);
    IF symFile # NIL THEN
        outSym := symFile.NewWriter(NIL);
        WriteLInt(outSym, SFtag)
    ELSE
        err(153)
    END
END NewSym;

PROCEDURE RegisterNewSym*;
    VAR res: INTEGER; name: Files.Name;
BEGIN
    IF symFile # NIL THEN
        name := ObjFName$;
        Kernel.MakeFileName(name, Kernel.symType);
        symFile.Register(name, Kernel.symType, Files.ask, res);
        symFile := NIL
    END
END RegisterNewSym;

PROCEDURE DeleteNewSym*;
BEGIN
    IF symFile # NIL THEN symFile.Close; symFile := NIL END
END DeleteNewSym;


(* write object file *)

PROCEDURE ObjW* (ch: SHORTCHAR);
BEGIN
    outObj.WriteByte(SHORT(ORD(ch)))
END ObjW;

PROCEDURE ObjWNum* (i: INTEGER);
BEGIN
    WriteNum(outObj, i)
END ObjWNum;

PROCEDURE ObjWInt (i: SHORTINT);
BEGIN
```

```
		outObj.WriteByte(SHORT(SHORT(i MOD 256)));
		outObj.WriteByte(SHORT(SHORT(i DIV 256)))
	END ObjWInt;

	PROCEDURE ObjWLInt* (i: INTEGER);
	BEGIN
		ObjWInt(SHORT(i MOD 65536));
		ObjWInt(SHORT(i DIV 65536))
	END ObjWLInt;

	PROCEDURE ObjWBytes* (IN bytes: ARRAY OF SHORTCHAR; n: INTEGER);
		TYPE P = POINTER TO ARRAY [untagged] 100000H OF BYTE;
		VAR p: P;
	BEGIN
		p := SYSTEM.VAL(P, SYSTEM.ADR(bytes));
		outObj.WriteBytes(p^, 0, n)
	END ObjWBytes;

	PROCEDURE ObjLen* (): INTEGER;
	BEGIN
		RETURN outObj.Pos()
	END ObjLen;

	PROCEDURE ObjSet* (pos: INTEGER);
	BEGIN
		outObj.SetPos(pos)
	END ObjSet;

	PROCEDURE NewObj* (IN modName: ARRAY OF SHORTCHAR);
		VAR res: INTEGER; loc: Files.Locator; dir: Files.Name;
	BEGIN
		errpos := 0;
		Kernel.Utf8ToString(modName, ObjFName, res); Kernel.SplitName(ObjFName, dir, ObjFName);
		loc := Files.dir.This(dir); loc := loc.This(codeDir);
		objFile := Files.dir.New(loc, Files.ask);
		IF objFile # NIL THEN
			outObj := objFile.NewWriter(NIL);
			WriteLInt(outObj, OFtag)
		ELSE
			err(153)
		END
	END NewObj;

	PROCEDURE RegisterObj*;
		VAR res: INTEGER; name: Files.Name;
	BEGIN
		IF objFile # NIL THEN
			name := ObjFName$;
			Kernel.MakeFileName(name, Kernel.objType);
			objFile.Register(name, Kernel.objType, Files.ask, res);
			objFile := NIL; outObj := NIL
		END
	END RegisterObj;

	PROCEDURE DeleteObj*;
	BEGIN
		IF objFile # NIL THEN objFile.Close; objFile := NIL END
	END DeleteObj;
	PROCEDURE InitHost;
		VAR test: SHORTINT; lo: SHORTCHAR;
	BEGIN
		test := 1; SYSTEM.GET(SYSTEM.ADR(test), lo); LEHost := lo = 1X;
		InfReal := SYSTEM.VAL(SHORTREAL, InfRealPat);
		MinReal32 := SYSTEM.VAL(SHORTREAL, MinReal32Pat);
		MaxReal32 := SYSTEM.VAL(SHORTREAL, MaxReal32Pat);
		MinReal64 := Compound(MinReal64PatL, MinReal64PatH);
		MaxReal64 := Compound(MaxReal64PatL, MaxReal64PatH)
	END InitHost;

BEGIN
	InitCrcTab;
	InitHost
END DevCPM.
```