Analgesia Database:
Console Library for PalmOS PDA

Version 0.95

J.M. van Schalkwyk

February 27, 2009

Contents

1 The C file: CONSOLE.c

This file and the subsequent files have been imported directly from the original C files, and largely haven't been reformatted in 'DogWagger' style, in other words, the LaTeX documentation is rudimentary.

```
/* =============================================================================
//                                    CONSOLE ROUTINES
//                                  For Palm SQL database
/* =============================================================================

//  OVERVIEW:
// for optimal debugging, it's great to have a 'console screen' we can write data
// This is that screen. It will allow us to write to console from ANY library, and
// also display that console at arbitrary points, or examine the 32K CONSOLE buffe
// after termination of the program on the Palm PDA.

// ------------------------
// 1. Fundamental routines:

// All we really need is:
//   1.1 The ability to write a text string to the console (may include fancy char
//       such as line feeds. We will use UNIX convention that line feed (0xA) is
//       the standard return character that designates a new line.
//       This generic write will be ConWrite(string, length)
//   1.2 The ability to read characters from the console:
//       1.2.1 Read the k most recent characters written (or until start, if lesse
//       1.2.2 Read the n most recent LINES
//       1.2.3 Read and remove the first k characters from the START of the buffer
//       1.2.4 Read and remove the first n LINES from the start of the buffer.
//       The above four might be something along the lines of:
//          ReadConChars (buf, bufsize)
//          ReadConLines (buf, bufsize, numlines)
//          KillConEarlyChars (buf, bufsize)
//          KillConEarlyLines (buf, bufsize, numlines)

//  Notes:
//  a. All line-reads are up to and including the end of line 0xA character.
//  b. We will implement the buffer as a 32K linear buffer file on the Palm PDA.

// ------------------------
// 2. Fancy routines:
// There are several 'frills' we *might* consider, but they will add little.
//    WE will NOT yet implement these, and there is a good argument against
//    implementing such routines, as they add to complexity.
//    2.1 Insert k characters at an arbitrary point within the console buffer
//    2.2 Delete k characters (ditto)
//    2.3 Write formatted string, given both string and another data type, e.g. a
```

```
//        floating point number. Best implemented in the caller, although we might
//        save space and bother by having such routines.
//    2.4 Locate an arbitrary search string within the entire console buffer, or
//        indeed, within a specified range
//        2.4.1 Locate the end of a particular line.

// ------------------------
// 3. Format of the CONSOLE
//    Total file size is 32128 bytes.
//    All numbers in the following are 4 byte numbers, with first 2 bytes = 0000
//
// Header: at start, 128 bytes [hmm?]
// +0  4 bytes: total file size (32128) All 4 byte numbers are big-endian (68K for
// +4  4 bytes: size of header
// +8  4 bytes: offset of start of data (earliest data)
// +C  4 bytes: offset of top of buffer (1st byte AFTER most recent byte)
// +0x10 .. 0x7F: reserved
// + 0x80.. 32127: actual data buffer.
//
// We will refer to offsets as follows: (see CONSOLE.h for formal definitions)
// +0   CONMAX
// +4   CONHDR
// +8   CONBOT
// +0xC CONTOP
//
// At present all character storage is ASCII, with no current provision for e.g. U
// When we move to unicode, we will be wanton, and store each character in 4 bytes
// all the pain of different character lengths at the cost of wasted space.
// [we might even use our UNIGLYPH].



/* ============================================================================


#include <SystemMgr.h>
#include <PalmOS.h>

#include "CONSOLE.h"
#include "../palmsql3A.h"



typedef struct {
   UInt16 refcount;
   DmOpenRef CONBUF;
   MemHandle CONHANDLE;
   Char *    CONPTR;
  } Console_globals;
```

```
/* ================================================================================
/* A. Some primitive functions. */

// --------------------------------------------------------------------------------
Err start (UInt16 refnum, SysLibTblEntryPtr entryP)
{
  extern void *jmptable ();
  entryP->dispatchTblP = (void *) jmptable;
  entryP->globalsP = NULL;
  return 0;
  }

// --------------------------------------------------------------------------------
Err nothing (UInt16 refnum) {
  return 0;
  }

// --------------------------------------------------------------------------------
// c_New: reserve memory:

MemPtr c_New ( Int16 memsize)
{
  MemHandle memH=0;
  MemPtr memP=0;
  memH = MemHandleNew(memsize);                 // *system* fx.
  if (! memH)
     { return 0;                                // fail.
     };
  memP = MemHandleLock(memH); // MHL1 unlikely to be an issue.
  if (! memP)                                   //
     { return 0;                                // fail.
     };
  return memP;                                  //pointer to locked new memory.
}

// --------------------------------------------------------------------------------
Int16 c_Delete (MemPtr memP)
{ if (MemPtrUnlock (memP) != errNone)
     { return 0;               // BadErrUnlockTempPtr;
     };
  if (MemPtrFree (memP) != errNone)
     { return 0;               // BadErrTempFreePtr;
     };
  return 1; // success
}
```

```
// ------------------------------------------------------------------------
// Copy from one pointer to another

Int16 xCopy (Char * dest, Char * xsrc, Int16 cnt)
{
  if (! dest)
     { return -1;
     };
  if (! xsrc)
     { return -2;
     };
  while (cnt > 0)                                  // permissible to copy NO bytes!
    { * dest++ = * xsrc++;
      cnt --;
    };
  return 0;     //ok
}


// ------------------------------------------------------------------------
// WriteInt32X is clumsy rtn from NUMERIC.c
//
Int16 WriteInt32X (Char * myptr, Int32 datum)
{ if (! myptr)
     {                                             // was: ERRmsg(ErNulInt16Write);
       return 0;                                   // attempt to write to null point
     };
  * (myptr+3) = (Char) datum & 0xFF;
  datum = datum >> 8;
  * (myptr+2) = (Char) datum & 0xFF;
  datum = datum >> 8;
  * (myptr+1) = (Char) datum & 0xFF;
  datum = datum >> 8;
  * (myptr) = (Char) datum;
  return 1;
}


// ------------------------------------------------------------------------
// Likewise for read:
//
Int32 ReadInt32X (Char * myptr)
{
  Int32 i;
  i =   0xFF & ((Int32) *(myptr+3));
  i |= (0xFF & ((Int32) *(myptr+2)))<<8;
  i |= (0xFF & ((Int32) *(myptr+1)))<<16;
  i |= (0xFF & ((Int32) *(myptr+0)))<<24;
  return i;
```

```
}


// ------------------------------------------------------------------------------
// Retreat. Given offset mypos in block pointed to by pointer P, move back until
//   either encounter 0xA (line feed) OR reach top of buffer, as indicated by
//   minpos:
// Return new offset in buffer (minimum is minpos)
//   returns -ve number if failed.

Int32 Retreat (Char * P, Int32 mypos, Int32 minpos)
{
  if (! P)
     { return minpos;  // fail
     };
  while ( mypos > minpos )
    { mypos --;                  // Do NOT detect 0xA at mypos on entry!
      if (* (P+mypos) == 0xA)
         { return mypos;   // return POINTING *TO* THE 0xA
         };
    };
  return minpos;
}


// ------------------------------------------------------------------------------
// Back n lines. Move back n lines in buffer.

Int32 BacknLines (Char * P, Int32 mypos, Int32 minpos, Int16 n)
{
  if (! P)
     { return minpos;  // fail. hmm.
     };
  while ((n > 0) && (mypos > minpos))            // if n<=0 at entry, leave mypos
    { mypos = Retreat(P, mypos, minpos);
      n --;
    };
    // there's an issue here:
    // if the character at P+mypos isn't an 0xA, we must be at the very start, and
    // we don't wish to skip past this first character;
    // similarly, if the character IS 0xA BUT n is > 0, we won't skip it:
    //
    if (  (*(P+mypos) == 0xA)
        &&(n == 0)
       )
       { mypos ++;                               // drop position by 1 to skip 0xA
       };
  return mypos;
}
```

```
// ------------------------------------------------------------------------------
LocalID c_DmFindDatabase (const Char *nameP)
{ // UInt16 cardNo is always zero, in our current schema.
  return DmFindDatabase (0, nameP);
}


// ------------------------------------------------------------------------------
Int16 c_DmCreateDatabase (const Char *nameP)
{ return (! DmCreateDatabase (0, nameP, DBCREATOR, DBTYPE, false)); // 0 signals C
}


// ------------------------------------------------------------------------------
DmOpenRef  c_DmOpenDatabase (LocalID dbID, UInt16 mode)
{
  return DmOpenDatabase (0, dbID, mode);
}


// ------------------------------------------------------------------------------
MemHandle  c_DmNewRecord (DmOpenRef dbP, UInt16 *atP, Int32 size)
{
  return DmNewRecord (dbP, atP, (UInt32)size);
}


// ------------------------------------------------------------------------------
MemHandle  c_DmGetRecord (DmOpenRef dbP, UInt16 index)
{
  return DmGetRecord (dbP, index);
}


// ------------------------------------------------------------------------------
Int16  c_DmWrite (void *recordP, Int32 offset, Char *srcP, Int32 bytes)
{
  if (bytes < 1)
     { return 0; // still 'succeed' if nothing written!
     };

  if (bytes > (CONSIZE-CONHEADSIZE))  // too much
     { return 1; // fail
     };
  if (offset+bytes > CONSIZE)
```

```
     { return 2; // fail
     };
  if (offset < 0)
     { return 3; // fail
     };
  return (DmWrite (recordP, (UInt32)offset, srcP, (UInt32)bytes));  // 0 signals S
}

// cannot write directly to file buffer hence the following: [HMM ???]
Int16 c_WriteInt32X (void * recordP, Int32 offset, Int32 datum)
{ Int32 xdatum;
  Char * Px;
  Px = (Char *) & xdatum;
  WriteInt32X ( Px, datum);  // coerce to big-endian!
  return c_DmWrite(recordP, offset, Px, 4); // ugly. 0=ok.

}



// ------------------------------------------------------------------------------
Int16 c_DmReleaseRecord (DmOpenRef dbP, UInt16 index, Boolean dirty)
{ return (! DmReleaseRecord (dbP, index, dirty)); // 0 = success!
}
```

Here's a v 0.95 addition, for database deletion.

```
Int16 c_DmDeleteDatabase (LocalID dbID)
{ Int16 ok;
  if (! dbID)
   { return 0; // fail
   };
  return(! DmDeleteDatabase (0, dbID)); // only for main card (0)
} // returns 0 on failure, nonzero on success.

// ------------------------------------------------------------------------------
Int16 c_MemHandleUnlock (MemHandle h)
{ return (! MemHandleUnlock (h) );  // 0 signals success thus negate
}


// ------------------------------------------------------------------------------
Int16 c_DmCloseDatabase (DmOpenRef dbP)
{ return (! DmCloseDatabase (dbP));  // 0 = success.
}

// ------------------------------------------------------------------------------
```

1.0.1 Fill routine

```
Int16 xFill (Char * p0, Int16 slen, Char c)
{   // okay, there is a faster way. Check out PalmOS dox.
  while (slen > 0)
    { * p0 = c;
      p0 ++;
      slen --;
    };
 return 1;           // success
}
```

1.0.2 Advance

Borrowed from *ScriptingLib.tex*, this routine scans a string for a particular character. It returns zero on failure, 1 after the desired character if it's located.

```
Int16 Advance (Char * myptr, Int16 limit, Char target)
{
  Int16 i = limit;
  while (  (limit > 0)
         &&(*myptr++ != target)
        )
        { limit --;
        };
  if (limit < 1)
     { return 0;        // fail
     };
  return 1+(i-limit);   // one AFTER character located
}

/* ================================================================================
/* B. BASIC FUNCTIONS. */

// ------------------------------------------------------------------------------
// open console:

Err CONSOLEOpen (UInt16 refnum)
{
  // here we must check whether file CONSOLE exists.
  // if not, create it.
  // We could also consider opening it and keeping handle
  // as global --- tricky but let's do so!

 SysLibTblEntryPtr entryP;
 Console_globals *gl;
 LocalID mydbid;
 UInt16 idx;
```

```
 UInt16* pIdx;
 Char * ErrorBufferName = "CONSOLE";
 Char * conhdr;  // pointer to new header we may create!
 Int16 cc;

 entryP = SysLibTblEntry (refnum);
 gl = entryP->globalsP;                      // access the globals
 if (!gl)
    {  /* We need to allocate space for the globals.  */
      gl = entryP->globalsP = MemPtrNew (sizeof (Console_globals)); // [*sys*]
      MemPtrSetOwner (gl, 0);
      gl->CONBUF = 0;
      gl->CONHANDLE = 0;
      gl->CONPTR = 0;
      gl->refcount = 0;
    };

 idx = 0;
 pIdx = &idx;

if (! gl->refcount)   // if very first instance, THEN ONLY:
   {
     mydbid = c_DmFindDatabase(ErrorBufferName);
     if (! mydbid)
        { if (! c_DmCreateDatabase(ErrorBufferName))
             { return CoErCannotCreate;              // fail
             };
          mydbid = c_DmFindDatabase(ErrorBufferName);
          if (! mydbid) { return CoErStillNotFound; }; // fail
          gl->CONBUF = c_DmOpenDatabase (mydbid, dmModeReadWrite);
          if (! gl->CONBUF) { return CoErCannotOpen; };  // fail
                 // create a new record (0) for newly made database:
          gl->CONHANDLE = c_DmNewRecord(gl->CONBUF, pIdx, CONSIZE);
          if (! gl->CONHANDLE) { return CoErNoHandle; };            // fail
          gl->CONPTR = (Char *) MemHandleLock(gl->CONHANDLE);       // MHL2

          // a. create 0x10 byte header buffer
          conhdr = c_New(0x10);

          // a clumsy hack: clear the whole console buffer,
          // to start off (1/4/2007):
          cc = (CONSIZE+CONHEADSIZE)/32;
          xFill(conhdr, 0x0E, 0x20); // 14 spaces
          *(conhdr) = 0xd;
          *(conhdr+1) = 0xa;
          while (cc >= 0)
            { cc --;
              c_DmWrite(gl->CONPTR, cc*16 , conhdr, 16); // fill up with blank lin
```

```
            };

          // here must initialise header ?!
          // b. fill it with relevant offsets
          WriteInt32X(conhdr+CONMAX,CONSIZE); // write total buffer size, hmm
          WriteInt32X(conhdr+CONHDR,CONHEADSIZE); // write header size
          WriteInt32X(conhdr+CONBOT,CONHEADSIZE); // write start = first byte
          WriteInt32X(conhdr+CONTOP,CONHEADSIZE); // write top = 1 after last used
          // [here might fill rest of header with zeroes, but we don't, at present
          // [rest of header is at present not accessed, and is dynamic so leave]
          // because boundaries are ok, above could use faster int32 write, but
          //  then must take endian issues into consideration with reading.
          // c. write it.
          c_DmWrite(gl->CONPTR, 0, conhdr, 0x10);  // at offset +0 write 0x10 byte
          // d. free temp header buffer
          c_Delete(conhdr);
        } else   // BUFFER ALREADY EXISTS:
        { gl->CONBUF = c_DmOpenDatabase (mydbid, dmModeReadWrite);
          if (! gl->CONBUF) { return CoErCannotOpen; };             // fail
          gl->CONHANDLE = c_DmGetRecord (gl->CONBUF, 0);    // record is #0.
          if (! gl->CONHANDLE) // fix is v~0.95: try to fix stubborn busy bits..
             { if (c_DmReleaseRecord(gl->CONBUF, 0, true)) // hack. 0=success!
                  { // failure .. try to delete (?!)
                    c_DmCloseDatabase(gl->CONBUF);
                    mydbid = c_DmFindDatabase(ErrorBufferName);
                    c_DmDeleteDatabase(mydbid); // delete without test
                    return CoErNoHandle; // fail
                  };  // else..
               gl->CONHANDLE = c_DmGetRecord(gl->CONBUF, 0);
               if (! gl->CONHANDLE)
                  { return CoErNoHandle; // fail
                  };
             };
          gl->CONPTR = (Char *) MemHandleLock(gl->CONHANDLE);  // MHL3
        };
   };

  gl->refcount++;
  return 0;
}



// ------------------------------------------------------------------------------
// close console.

Err CONSOLEClose (UInt16 refnum, UInt16 *numappsP)
 {
   SysLibTblEntryPtr entryP;
```

```
   Console_globals *gl;
   entryP = SysLibTblEntry (refnum);
   gl = entryP->globalsP;
   if (!gl)
    { return CoErCloseNotOpen;                  // we're not open
    };

  *numappsP = --gl->refcount;                   // predecrement count, store.
  if ((gl->refcount) > 0)                       // if nonzero
    { return 1;                                 // signal 'others still talking'
    };

  if (! c_MemHandleUnlock(gl->CONHANDLE))
     { return CoErNotUnlock;
     };
  if (! c_DmReleaseRecord(gl->CONBUF, 0, true))
     { return CoErNotRelease;
     };
  if (! c_DmCloseDatabase(gl->CONBUF))
     { return CoErNotCloseDB;
     };

   /* Clean up.  */
   MemChunkFree (entryP->globalsP);
   entryP->globalsP = NULL;

  return 0;
 }
```

1.1 SubScroll

Given pointer to start of buffer, clip out `howmany` bytes from the start of the buffer and shift everything up. Nasty and slow; should simply be replaced by a cyclical write to the buffer.[1]

The following section *was* in error. You cannot directly manipulate file memory, so we caused a nasty crash. The fix is to simply write using the relevant, wrapped PalmOS functions. Fortunately the overlapping move works!

```
Int16 SubScroll (Char * P, Int32 howmany)
 { Int32 hdrsize;
   Int32 contop;
   Int32 tomove;
```

[1] We will then need to alter the extraction routines appropriately.

```
  hdrsize =  CONHEADSIZE;
  contop =  ReadInt32X(P+CONTOP);         // get current top
  tomove = contop - (hdrsize + howmany);
  if ((tomove < 0) || (tomove > (CONSIZE - CONHEADSIZE)))
     { c_WriteInt32X(P, CONTOP, hdrsize); // if stuffup, just CLEAR!
     } else
     { c_DmWrite(P, hdrsize, P+howmany+hdrsize, tomove);
       c_WriteInt32X(P, CONTOP, tomove+hdrsize);
     };
  return 0;   // ok
 }
```

We obtain the size of the header in the buffer (`hdrsize`), and the offset of the current top of the buffer (which includes the header size). If we're trying to clear too much, just clear everything and return; otherwise clip out `howmany` bytes from the front of the buffer, moving the upper text chunk over the bytes clipped out, and decreasing the pointer to the buffer top commensurately.

```
// given string, write it to the console buffer.

Int16 ConWrite (UInt16 refnum, Char * strP, Int16 strL)
{
  SysLibTblEntryPtr entryP;
  Console_globals *gl;

  Int32 conoff;
  Int32 conmax;
  Char * conP;  // pointer to console buffer
  Int32 overrun;
  Int32 fail2;
  Int16 fail = 0;
  Int32 strlen = strL; // hack.

  Int16 adv = 0;

  +OPTIONAL
  return 0; // OPTION = DISABLE ALL WRITING TO CONSOLE!! 19/3/2006
  -OPTIONAL

  if (strlen <= 0)
     { if (strlen == 0)
          { return 0;  // success, despite writing nothing!
          };
       return -1;  // fail. Might extend this to writing of say 32K blocks
                   // in larger implementation!
     };
```

```
entryP = SysLibTblEntry (refnum);
gl = entryP->globalsP;
if (!gl)
 { return CoErCloseNotOpen;                 // we're not open
 };

// next, read current offset, write to it, and then update the offset:
conP = gl->CONPTR;
conoff =  ReadInt32X(conP+CONTOP);     // get current top
conmax =  ReadInt32X(conP+CONMAX);     // get total size (including header)


overrun = (conoff + strlen) - conmax;         // find overrun space at top
if (overrun > 0)                              // if insufficient space..
   {
     SubScroll(conP, strlen+1024); // simple
     // we add 1024 to improve performance
     conoff =  ReadInt32X(conP+CONTOP); // get NEW top
   };
```

In the following, rather than simply writing to the 'console', we test for the presence of a backslash character. If we then find \n, we translate this as a line feed (hexadecimal 0xA). We leave other backslash-prepended characters unchanged, and (at least for now) don't render a double backslash as a single one!

```
while (  (adv = Advance (strP, strlen, '\\') )  ) // ugh!
      {
        c_DmWrite(conP, conoff, strP, adv-1); // don't include '\'
        if (*(strP+adv) == 'n')
           { c_DmWrite(conP, conoff+adv-1, "\n", 1); // write LF
             strP ++;
             strlen --;
           } else
           { // here might test for other \chars!
             c_DmWrite(conP, conoff+adv-1, "\\", 1); // write backslash
           };
        strlen -= adv;
        strP += adv;
        conoff += adv;
      };

c_DmWrite(conP, conoff, strP, strlen);        // write to buffer
conoff += strlen;

fail2 = c_WriteInt32X(conP, CONTOP, conoff);     // 0 = success, !0 = fail
if (fail2)
   { return fail2; // CoErOffWrite;
```

```
      };
  return fail;                                  // 0 = success
}


// ------------------------------------------------------------------------------
// Read given number of characters from the console buffer.
// Does NOT alter contents of buffer in any way.
//   the 'mode' argument is one of:
//      0 = normal. Simply read last 'strlen' characters from the *top* of the buf
//     any positive number k (non 0):
//       read characters starting from  offset k in buffer!
//     -n: move n lines back, then fetch max of strlen characters!
//     -j (j>1) error
//
// note that you are allowed to read zero characters from the buffer.
//
// RETURN VALUE:
// RETURNS *NUMBER OF CHARACTERS PLACED IN BUFFER*
//   if not at start of buffer: returns n = characters read;
//   IF at start of buffer, returns -n.
//   This odd arrangement is to allow detection of 'start of buffer' condition
//     WITHOUT other flags etc.
//   'absolute value of number returned' is always number of bytes written!
// NOTE: IDEALLY SHOULD WRAP AROUND IN BUFFER!

Int16 ConRead (UInt16 refnum, Char * dest, Int16 strL, Int16 mode)
{
  SysLibTblEntryPtr entryP;
  Console_globals *gl;
  Char * P;
  Int32 strlen;
  Int32 constart;
  Int32 contop;
  Int32 conpos;

strlen = strL;

  if (strlen < 0)  // nonsense
     { return 0;
     };
  if (strlen > (CONSIZE-CONHEADSIZE))  // too much demanded!
     { return 0;                                  // [??? check me]
     };

  entryP = SysLibTblEntry (refnum);
  gl = entryP->globalsP;                       // access the globals
  if (!gl)
```

```
    { return 0;
    };
  P = gl->CONPTR;
  constart = ReadInt32X(P+CONHDR);       // get header size
  contop =  ReadInt32X(P+CONTOP);         // get current top

  if (! mode )             // NORMAL MODE is zero
    { conpos = contop - strlen;               // point to first of requisite char
    } else                 // ABNORMAL MODE:
    { if (mode < 0)
        { // if negative (ie -k), move k lines up:
          conpos = BacknLines (P, contop, constart, -(mode));
          // BacknLines will move to AFTER the 0xA unless at top!
        } else                                 // mode now specifies OFFSET!
        { conpos = constart + mode;            // add it
        };
    };

  // check sensibility:
  if (conpos < constart)                       // don't transgress into header
    { conpos = constart;
    };
  if (conpos > contop)                         // position above top??
    { strlen = 0;   // force no copy!
    };

  if (strlen > (contop - conpos))              // asking too much
    { strlen = contop - conpos;
    };

  xCopy(dest, P+conpos, strlen);

  if (conpos == constart)
    { return -strlen;                          // PECULIARITY: see documentation
    };
  return strlen;                              // return number of chars fetched
}



/*   ============================================================================
/*                                        THE END
/*   ============================================================================
```

2 Header file: CONSOLE.h

```
#ifndef CONSOLE_H
#define CONSOLE_H

#include <LibTraps.h>
#include <FloatMgr.h>

#ifndef CONSOLE_TRAP
#define CONSOLE_TRAP(trapno)  SYS_TRAP(trapno)
#endif

#define CONHEADSIZE 128
#define CONSIZE 32000+CONHEADSIZE

#define CONMAX 0
#define CONHDR 4
#define CONBOT 8
#define CONTOP 0x0C


#define CONSOLEWrite            sysLibTrapCustom+0
#define CONSOLERead             sysLibTrapCustom+1


Err CONSOLEOpen (UInt16 refNum)
    CONSOLE_TRAP(sysLibTrapOpen);

Err CONSOLEClose (UInt16 refNum, UInt16 *numappsP)
    CONSOLE_TRAP(sysLibTrapClose);

Int16 ConWrite (UInt16 refnum, Char * strg, Int16 strlen)
    CONSOLE_TRAP(CONSOLEWrite);

Int16 ConRead (UInt16 refnum, Char * dest, Int16 strlen, Int16 mode)
    CONSOLE_TRAP(CONSOLERead);



#define CoErCannotCreate        15001
#define CoErStillNotFound       15002
#define CoErCannotOpen          15003
#define CoErNoHandle            15004
#define CoErCloseNotOpen        15005
#define CoErNotUnlock           15006
#define CoErNotRelease          15007
#define CoErNotCloseDB          15008
#define CoErClipTooMany         15009
// #define CoErOffWrite            15010
```

```
#endif
```

3 The Makefile

```
LIBPATH = c:/palmdev/sdk-4
CREATOR = JxVS
LIBPATH = c:/palmdev/sdk-4
VERSION = 1

CC = m68k-palmos-gcc -Wall -g -O2
AS = m68k-palmos-as

all: CONSOLE-syslib.prc

CONSOLE-syslib.prc: CONSOLE.def CONSOLE
build-prc -o $@ CONSOLE.def CONSOLE
ls -l *.prc

CONSOLE_objs = CONSOLE.o CONSOLE-dispatch.o

CONSOLE: $(CONSOLE_objs) Makefile
$(CC) -shared -nostartfiles -nostdlib -o $@ $(CONSOLE_objs) -lnfm -lgcc
m68k-palmos-objdump --section-headers CONSOLE

CONSOLE.o: CONSOLE.c CONSOLE.h

CONSOLE-dispatch.o: CONSOLE-dispatch.s

CONSOLE-dispatch.s: CONSOLE.def
m68k-palmos-stubgen CONSOLE.def

clean:
rm -f *.o *.prc *-dispatch.? CONSOLE
```

4 The DEF file: CONSOLE.def

```
syslib { "CONSOLE Library" CnLi }

export {
  CONSOLEOpen CONSOLEClose nothing nothing
  ConWrite ConRead
  }
```

5 Change Log

From version 0.95, we introduce a change log.

5.1 Version 0.95

1. Due to an OS error (Palm OS 5.4.9), if the program is interrupted while open and a soft reset occurs, busy flags are left set within open records. This can be devastating. This problem is discussed in *CProgMain.tex*; we use a similar fix here. The main area of concern is ‘MHL3’ above.