The Interactive Engineer

ie

Volume 5, Number 4, July/August 1996

Editorial

Are you back from the holidays? Maybe you are,but you're probably still in the mood of palm beaches and ice cream. We know it isn't easy get back behind your desk or your terminal. Or worse,behind your terminal on your desk.

Holiday memories apart, we received some interesting comments from long-standing members of the international CD-i scene. They discuss some issues that were brought up in the previous Interactive Engineer. Hats off and a box of irresistible Belgian chocolates for them. We are always happy to distribute your knowledge through the Interactive Engineer to improve CD-i programming. So keep those comments coming in.

This month's Interactive Engineer gives some tips on how to use the CD-i player's RS232 port for data reading,have a look at the sample code starting on page 2. You can save time by mailing the PIMC mail server at mailserv@pimc.be and request a soft copy of the code.

This issue also includes a preview of the long-awaited CD-i Internet Toolkit. Meanwhile,enjoy this issue of Interactive Engineer. We are looking forward to receiving your reactions.

IE editorial staff

Don't forget that when your ideas are published in IE, we will send you, as reward, a box of authentic Belgian chocolates.

■ RS232 Sample code

If your application has to read data through the serial port,the listing presented here will help you to implement the right C-code.

This program performs the necessary initialisation for Balboa and the RS232 receiver. Then all received bytes are written to the terminal.

This program is only an example, you'll probably need some tuning to fit all this in your application.

```
/* Include files */
#include "general.h"
#include <sgm.h>
#include <sgstat.h>

/* Function prototypes */
intfInit();
void   fImageLoaded();
void   fMyError();
void   fMyQuit();

/* Constant Definitions */
#define MAX_INTERNAL_BUF  1
#define MAX_DSEG    2

/* Global Variables */
VS *pVs;      /* Pointer to Balboa Video Screen. */
PICTURE       *pPicCA;     /* Pointer to CLUT picture in plane A */
intlT2Path;  /* Path for the /t2 device */

/* ----------------------------------------------------------------------------
 * Function Name:  fRsCommand
 * Description:    Received a command from the RS232 communication.
 * Do something useful with it.
 * Parameters:     lContext,pParam.
 * Returns:  Nothing.
 */

void fRsCommand(lContext,pParam)
intlContext;
void   *pParam;
{
  STATUS(ST_APPL, ST_ENTRY, lContext, "fRsCommand()");
  printf("\t%c\n", (char)lContext);
}
/* ----------------------------------------------------------------------------
 * Function Name:  fRsReceived
 * Description:    Received a signal from the RS232 communication.
 * Parameters:     lContext,pParam.
 * Returns:  Nothing.
 */

void fRsReceived(lContext,pParam)
intlContext;
void   *pParam;
{
  int  i, lNumBytes;
   cRsBuffer[128];
  short      sSigCode;

  STATUS(ST_APPL, ST_ENTRY, 0, "fRsReceived()");

  /* keep the signal number for re-arming*/
  sSigCode = (short)pParam;
  /*
   * read all bytes from the CD-i player's buffer. Never read more bytes than
```

```
    * indicated by the driver, else You go to sleep
    */
   lNumBytes = _gs_rdy( lT2Path);
   if (lNumBytes == SYSERR)
   {
    fMyError("_gs_rdy()");
   }
   if ((i = read( lT2Path, cRsBuffer, lNumBytes)) != lNumBytes)
   {
    STATUS(ST_APPL, ST_ERROR, lNumBytes - i, "Not enough bytes read, missed ");
   }

   /* do something with the received bytes */
   for (i = 0; i < lNumBytes; i++)
   {
    dispatch_function(cRsBuffer[i], fRsCommand, NULL);
   }
   /* re-arm the signals */
   if ( _ss_ssig(lT2Path, sSigCode) == SYSERR)
   {
    fMyError(" _ss_ssig()");
   }
 }

 /* ------------------------------------------------------------------------
  * Function Name:   fExitRs
  * Description:     This function closes the RS232 communication.
  * Parameters:      none.
  * Returns:  Nothing.
  */
 int fExitRs()
 {
   STATUS(ST_APPL, ST_ENTRY, 0, "fExitRs()");

   /* detach /t2, this enables a decent exit */
   if (detach("/t2", S_IREAD | S_IWRITE) == SYSERR)
   {
    fMyError("detach(/t2)");
   }
   return OK;
 }

 /* ------------------------------------------------------------------------
  * Function Name:   fInitRsReceive
  * Description:     This function initializes the RS232 communication
  * for receiving commands.
  * Parameters:      none.
  * Returns:  Nothing.
  */
 int fInitRsReceive()
 {
 struct sgbuf sOptBuffer;
 intlRsClass, lRsSig;

   STATUS(ST_APPL, ST_ENTRY, 0, "fInitRsReceive()");

   /* setup the signal handler */
   lRsClass = sgm_new_class(1, BP_MEM_PLANEA);
   if (lRsClass == SYSERR)
   {
    fMyError("sgm_new_signal()");
   }
   lRsSig = SGM_SIGNAL(lRsClass, 0);
   if (sgm_enable(lRsSig, fRsReceived, (void *)lRsSig, DISPATCHED) != OK)
   {
    fMyError("sgm_enable()");
   }
   /* attach /t2, this enables a decent exit */
   if (attach("/t2", S_IREAD | S_IWRITE) == SYSERR)
  {
   fMyError("attach(/t2)");
  }
  /* open the path to /t2 */
  lT2Path = open("/t2", S_IREAD | S_IWRITE);
  if (lT2Path == SYSERR)
  {
   fMyError("open(/t2)");
  }
  /* set the correct transmission parameters */
  if (_gs_opt(lT2Path, &sOptBuffer) == SYSERR)
  {
   fMyError("_gs_opt()");
  }
  /* first set the buffer to zero */
  memset(&sOptBuffer, 0, sizeof(sOptBuffer));
  /* fill in parity, databits, stopbits, baudrate */
  /* details can be found in the OS-9 technical I/O manual */
  sOptBuffer._sgm._sgs._sgs_parity = 0x00; /* no parity, 8 databits, 1 stop */
  sOptBuffer._sgm._sgs._sgs_baud = 0x0e;      /* 9600 baud, 0x07 for 1200 baud */
  /* and write to driver */
  if (_ss_opt(lT2Path, &sOptBuffer) == SYSERR)
  {
   fMyError("_ss_opt()");
  }
  /* arm the signals */
  if ( _ss_ssig(lT2Path, lRsSig) == SYSERR)
  {
   fMyError(" _ss_ssig()");
  }

  STATUS(ST_APPL, ST_EXIT, 0, "fInitRsReceive() OK");
  return OK;
}

/* ----------------------------------------------------------------------------
 * Function Name:  fInit
 * Description:    This function initializes some Balboa managers and
 * starts to load the first screen.
 * Parameters:     Argc   Argument count      (not used)
 * Argv      Argument list      (not used)
 * Returns:  Nothing.
 */
int fInit(argc, argv)
int argc;
char *argv[];
{
  STATUS(ST_APPL, ST_ENTRY, 0, "fInit()");
  /* Initialize Balboa Signal Manager */
  sgm_init();
  /* Initialise Balboa Timer Manager - use 1 timer*/
  tmr_init(1, BP_MEM_PLANEA);
  /* Initialise the receiving via RS232 */
  fInitRsReceive();
  /* Install a quit-handler */
  if (dispatch_atquit(fMyQuit,NULL) == SYSERR)
  {
   fMyError("dispatch_atquit()");
  }
  /* Initialize video environment */
  if (vs_init(BP_MEM_DONTCARE, MAX_INTERNAL_BUF) == SYSERR)
  {
   fMyError("vs_init()");
  }
  pVs = vs_open(LCT_DOUBLE, MAX_DSEG, BP_MEM_PLANEA);
  if (pVs==NULL)
  {
   fMyError("vs_open()");
  }
  /* Initialize the CLUT manager */
  if (cl_init(pVs,BP_MEM_PLANEB,0)==SYSERR)
  {
   fMyError("cl_init()");
  }
  /* Create a picture for a full-screen CLUT image */
  pPicCA = pi_create(PLANE_A,D_CLUT7,TO_HIRES(PAL_WIDTH),TO_HIRES(PAL_HEIGHT),
  PAL_SECTORS*F2_BYTES,0);
  if (pPicCA == NULL)
  {
   fMyError("pi_create()");
  }
  /* Allocate memory to load the clut-table into: one sector */
  pPicCA->pi_clut = (CLUT *)bp_allocate(F1_BYTES,BP_MEM_PLANEA);
  if (pPicCA->pi_clut == NULL)
  {
   fMyError("bp_allocate() of pi_clut");
  }
  /* We won't load a picture here, but keep program structure */
  pi_clear(pPicCA, 0);
  pPicCA->pi_clut->cl_offset = 0;
  pPicCA->pi_clut->cl_count = 1;
  pPicCA->pi_clut->cl_vals[0] = 0x80;
  pPicCA->pi_clut->cl_vals[1] = 0x10;
  pPicCA->pi_clut->cl_vals[2] = 0x10;
  dispatch_function(0, fImageLoaded, NULL);
}

/* ---------------------------------------------------------------------
 * Function Name:  fWaitingDone
 * Description:    The waiting period is over.
 * Parameters:     lContext,pParam.
 * Returns:  Nothing.
 */
void fWaitingDone(lContext,pParam)
intlContext;
void   *pParam;
{
  STATUS(ST_APPL, ST_ENTRY, 0, "fWaitingDone()");
  dispatch_quit(0);
}

/* ---------------------------------------------------------------------
 * Function Name:  fImageLoaded
 * Description:    This function initializes the picture list and
 *displays the image.
 * Parameters:     lContext     Balboa Callback context
 *pParam     Optional User Parameter
 * Returns:  Nothing.
 */
void fImageLoaded(lContext,pParam)
intlContext;
void   *pParam;
{
  STATUS(ST_APPL, ST_ENTRY, 0, "fImageLoaded()");
  /* Update the video processor's clut tables */
  if (cl_exec(pVs, PLANE_A, pPicCA->pi_clut) == SYSERR)
  {
   fMyError("cl_exec()");
  }

  /* Initialize the picture stack: pPicCA in front */
  if (pi_front(pVs,pPicCA) == SYSERR)
  {
   fMyError("pi_front()");
  }

  /* Update the screen to show the picture just loaded */
  if (vs_update(pVs,NON_INTERLACE,vm_display_type,NULL,NULL)==SYSERR)
  {
   fMyError("vs_update()");
  }

  /* Show the screen */
  if (vs_show(pVs)==SYSERR)
  {
   fMyError("vs_show()");
  }

  /* Wait a bit - one minute. In the mean time we'll receive bytes through
  the rs232 port, and print them. */
  if (tmr_after(1 * 60 * 100, fWaitingDone, NULL, DISPATCHED) == SYSERR)
  {
   fMyError("tmr_after()");
  }
}

/* ---------------------------------------------------------------------------
 * Function Name:  fMyError
 * Description:    This function handles errors.
 * Parameters:     pMessage     Message to display.
 * Returns:  Nothing.
 */
void fMyError(pMessage)
char   *pMessage;
{
  STATUS(ST_APPL,ST_ERROR,errno,pMessage);
  dispatch_quit(errno);
  dispatch_continue();
}

/* ---------------------------------------------------------------------------
 * Function Name:  fMyQuit
 * Description:    This function handles title exit.
 * Parameters:     lContext     Balboa Callback context
 * pParam     Optional User Parameter
 * Returns:  Nothing.
 */
void fMyQuit(lContext,pParam)
intlContext;
void   *pParam;
{
  STATUS(ST_APPL,ST_ENTRY,0,"fMyQuit()");
  fExitRs();
  vs_close(pVs);
  vs_finish();
  STATUS_FLUSH();
  STATUS_CLOSE();
  exit(OK);
}

/* ---------------------------------------------------------------------------
 * Function Name:  main
 * Description:    Main entry point of application
 * Parameters:     Argc   Argument count
 * Argv      Argument list
 * Returns:  Never returns
 */
main(argc, argv)
intargc;
char * argv[];
{
  /*
   * Initialize the status manager to show all types of messages from
   * all managers.
   */
  STATUS_INIT(stdout, ST_MGR_ALL, ST_TYP_ALL, 0);
  /*
   * Start the dispatcher. The first program to be dispatched must be
   * fInit.
   */
  dispatch_loop(fInit, argc, argv);
}


/* ---------------------------------------------------------------------------
 * Filename: general.h
 * Author:   PIMC
 * Description: General includes and definitions for any Balboa CD-i title.
 * Use of Digital Video and Balboa version 2.1 is assumed.
 * If no Digital Video is used, the include files mv.h and
 * ma.h may be omitted.
 */
#ifndef _GENERAL_H
#define _GENERAL_H

/* ------------------------- OS-9 INCLUDES ------------------------- */
#include <stdio.h>
#include <modes.h>
#include <csd.h>
#include <ucm.h>
#include <mv.h>
#include <ma.h>

/* ------------------------- BALBOA INCLUDES ------------------------- */
#include <balboa.h>
#include <bp_error.h>
#include <bp_mem.h>
#include <controls.h>
#include <pm_low.h>
#include <sgm.h>
#include <status.h>
#include <vm_defs.h>
#include <vm_pic.h>
#include <vm_txt.h>
#include <vm_video.h>
#include <vm_vs.h>

/* ------------------------- CONSTANT DEFINITIONS ------------------------- */
#define PAL_WIDTH   384
#define PAL_HEIGHT  280
#define NTSC_WIDTH  384
#define NTSC_HEIGHT 240
/* Size in sectors of a full-screen dyuv image */
#define PAL_SECTORS 47
#define NTSC_SECTORS      40
/* Number of data bytes in a Form 1 or Form 2 sector */
#define F1_BYTES    2048
#define F2_BYTES    2324
/* Dummy value (Not Applicable), used as int equivalent of pointer NULL */
#define NA   -1
/* Maximum length of any path name (more strict than neccessary) */
#define MAX_LEN_PATH      80

/* ------------------------- MACRO DEFINITIONS ------------------------- */
/* Convert a low-res value to high-res */
#define TO_HIRES(x) (x<<1)
/* Convert a high-res value to low-res */
#define TO_LORES(x) (x>>1)
/* Convert a single channel number to it's corresponding bitmask */
#define CH_MASK(x)  (1<<x)
/* Convert a single value to it's corresponding channel number */
#define CHANNEL(x)  (x)
/* Convert a number of bytes to the corresponding number of FORM2 sectors */
#define F2_SECTORS(x)     (((F2_BYTES - ((x) % F2_BYTES)) + (x)) / F2_BYTES)
```

```
/* ---------------------- USER DEFINED ERROR CODES ---------------------- */
/* Define a base error number analogous to Balboa. Balboa reserves the
  range 042:000 through 099:000, so we will use 041:000 - 041:999 */
#define ERR_BASE    (41<<8)
#define ERR_INV_PARAM     (ERR_BASE+1) /* Invalid parameter */
#define ERR_INTERNAL      (ERR_BASE+2) /* Internal error (programming bug) */
#define ERR_NOT_INIT      (ERR_BASE+3) /* Not initialized */
#define ERR_DUP_INIT      (ERR_BASE+4) /* Duplicate initialization */

#endif /* _GENERAL_H */
```

Rudi Verslegers

■ Direct Line Control Table access

In the IE from March/July 1996,Neil Kenneally describes a problem with directly accessing the system line control tables (LCTs),caused by incorrect assumptions about the memory addresses of these tables.

If you really need direct access to the system LCTs, you can safely find the address of line 1 of a Normal/Double Resolution LCT with the following function:

```
char *lct_line_1 (vidpath, plane, lctid)

/* path to video device */
int vidpath;

/* plane for LCT, either PA or PB */
int plane;

/* LCT identifier returned by dc_crlct*/
int lctid;
{
  int lctdummy, lnkinstr;

  /* create a dummy LCT in the same plane
     */
  lctdummy=dc_crlct(vidpath, plane, 2, 0);

  /* link this dummy LCT to line 1 of our
     parameter LCT */
  dc_llnk(vidpath, lctdummy, 1, lctid, 1);

  /* read the link instruction written by
     dc_llnk */
  lnkinstr=dc_rdli(vidpath, lctdummy, 1,
     7);

  /* delete the dummy LCT */
  dc_dllct(vidpath, lctdummy);

  /* the address is in the lower 24 bits
     of the link instr */
  return (char *) (lnkinstr & 0x00ffffff);
}
```

Each system LCT consists of 64 bytes per line, of which the first 32 contain the 8 LCT instructions. The last 32 bytes should be left alone (although some player implementations execute LCT instructions from there too).

You should not directly modify the LCT line (usually line zero) to which the FCT is linked (with dc_flnk),because the hardware may or may not actually read that line from this memory location (many player implementations copy the line into the system-maintained shadow FCT).

There is no corresponding way to find the address of the system FCTs. CD-RTOS may keep its own shadow copy of the FCT for the hardware to read,and there is no CD-RTOS call to access (directly or indirectly) the address of any FCT.

The technique described above works on all current players and is likely to remain so,because many discs currently in the market use this technique or some variant of it.

The above technique,however, is slightly beyond the Green Book:

- GB V.4.5.1 (Figure V.45) gives the format of the link instruction and states the location where it will be written by DC_LLnk. Many player implementations use a different Code value then given in this table ($30 instead of $20),but this is not exploited by our technique.
- GB VII.7.5.2.2.4 states that each LTC line used 64 bytes.

Luc Rooijakkers, SPC Vision

■ More on Balboa and soundmaps.

About soundmap memory substitution.

In the IE from March/July 1996,Rudi Verslegers describes the use of soundmaps in Balboa. He also describes a special technique that can be used to circumvent difficulties which can occur when soundmaps are created with implicit malloc's. The technique involves creating a small soundmap (typically one soundgroup) and then changing the SMD_SMAddr and SMD_SMSize fields of the soundmap descriptor (described in Green Book chapter VII.2.2.4.1),thus substituting your own memory for the system-allocated chunk.

First,the use of the term 'malloc'is not entirely correct. CD-RTOS allocates the soundmap memory with the F$SRqMem system call, not with the application's malloc function. The difficulties that can occur are described in PIM Technical Note #87,Memory Allocation in CD-RTOS.

Secondly, you should save the old values of SMD_SMAddr and SMD_SMSize and restore them before application exit,because CD-RTOS uses these values to free the allocated soundmap memory.

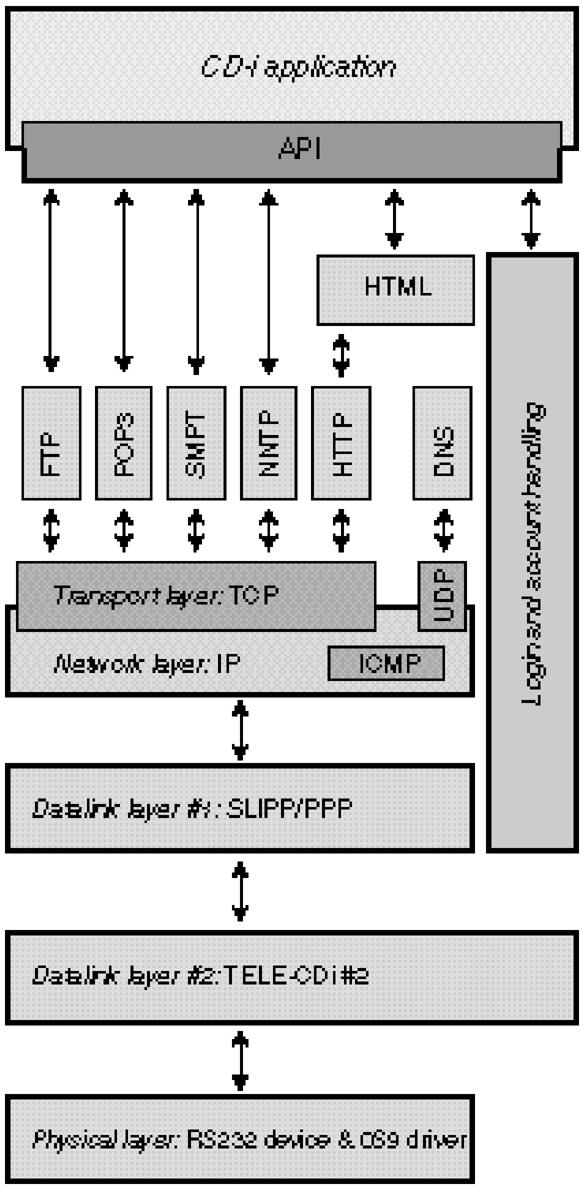
Finally, it may be a good idea to save,modify and restore the SMD_NoGrps field of the soundmap descriptor as well.

Luc Rooijakkers, SPC Vision

CD-i Internet toolkit

The Internet toolkit allows you to embed Internet functions into your applications without having to deal with the low level protocols, complex administration or other complicated matters. The Toolkit consists of an Applications Programmers Interface (API) with C-functions, including functions to build WEB Browsers,E-mail functions,FTP and News Groups.

The following figure represents a schematic overview of the CD-i communication model:


CD-i application
API
HTML
FTP
POP3
SMPT
NNTP
HTTP
DNS
Login and account handling
Transport layer: TCP
UDP
Network layer: IP
ICMP
Datalink layer #1: SLIPP/PPP
Datalink layer #2: TELE-CDi #2
Physical layer: RS232 device & OS9 driver

• Application Level Protocols

FTP:stands for 'File Transfer Protocol'. This application protocol makes it possible to transfer files across the network.

HTTP:stands for 'HyperText Transfer Protocol'. This application protocol provides a fast and light mechanism for transmitting files over a TCP/IP link. Because of the two advantages,mentioned above,it is mostly used for transferring hypertext documents (HTML). The location of such a document is given by its Uniform Resource Locators (URL's).

SMPT: stands for 'Simple Mail Transfer Protocol'. This application protocol provides a reliable and efficient mechanism for transmitting mail.

NNTP:stands for 'Network News Transfer Protocol'. This application protocol specifies distribution,inquiry, retrieval and posting of news articles.

POP3:stands for 'Post Office Protocol version 3'. This application protocol allows a workstation (in this case the CD-i) to retrieve mail that a server is holding for it.

HTML:stands for the 'HyperText Markup Language'. This application protocol is used to define the structure of hypertext documents on the World Wide Web.

• Transport layer

The 'Transmission Control Protocol'(TCP) is responsible for breaking up messages into data blocks at one end and reassembling them at the other end,ordering the received datablocks and re-sending lost datablocks.

The 'User Datagram Protocol'(UDP) is similar to TCP except that it doesn't split up messages into blocks and doesn't resent missing blocks. This protocol is faster but less reliable then TCP. Domain Name Servers (DNS) use this protocol to look after IP addresses.

• Network layer

The 'Internet Protocol'(IP) is designed to connect a large number of networks together, with the current 'Internet'as result. It transmits blocks of data (datagrams) from a source to a destination where both are identified by a fixed length address -the famous IP address. It's main task is routing information throughout the network.

The 'Internet Control Message Protocol' (ICMP) is part of the Network Layer (IP) and transmits error and control messages to the TCP/IP layer.

• Datalink Layer #1

The 'Serial Line Internet Protocol'(SLIP) and 'Point-To-Point'(PPP) protocol define the mechanism by which asynchronous serial connections are modified in order to send IP information. This layer transforms the communication channel into an error free line by recovering lost,double or damaged frames.

• Datalink Layer #2

Tele-CD-i send and receive characters to and from a modem connected to the CD-i's serial port. Tele-CD-i is a modem toolkit with it's own Application Programmers Interface which interfaces with the Internet toolkit through a data module,created by Tele-CD-i when initialized.

• Physical layer

This layer holds the serial device driver and the OS-9 serial manager. The application controls the asynchronous serial communication (RS232) through these drivers. This multipurpose serial connection is able to sent and receive up to 19200 bits per second.

The list below gives an overvieuw of the available functions in the Internet Toolkit. The format of the functions is defined as
ink_XXX_<name>
where:

XXX specifies the class of the function

<name> specifies the function name

The following classes are defined:

CON	Connect Class;
WWW	Web Class;
HTM	Display-HTML-Pages Class;
MAI	Mail Class;
FTP	File Transfer Class;
NWS	News Class;
AAG	Account Generation Class;
CTR	Control & Support Class.

• Connect & Login functions

```
intink_CON_ConnectLogin         (int baudrate, int databits, int stopbits,
                                 int parity, int framingmode, char *userid,
                                       char *password, int InactivitySignal,
                                 int InactivityTimeout)
int ink_CON_Disconnect          (void)
int ink_CON_CheckConnectState   (void)
int ink_CON_GetMyIP             (void)
```

• WEB functions

```
int ink_WWW_GetPage             (char *pcUrl, char *pcUserid, char *pcPassword,
                                 int*piPageID)
int ink_WWW_GetImage            (char *pcUrl, int *piImageID)
int ink_WWW_PostPage            (char *pcUrl, char *pcUserid, char *pcPassword,
                                 int*piPageID)
```

•HTML functions

```
int ink_HTM_CreateWindow        (WinParam *psWin)
int ink_HTM_DeleteWindow        (int iWinID)
int ink_HTM_SetWindowProperties (int iWinID, WinParamsub *psWin)
int ink_HTM_CheckHotspot        (int iX, int iY, char *pcUrl, int iType)
int ink_HTM_ShowHTML            (int iAction, int iPageID, int iWinId)
int ink_HTM_DecodeImage         (int iImageID)
int ink_HTM_CopyPage            (int iPageID, char *pcRawData)
int ink_HTM_GetPageInfo         (int iPageID, PagaData *psPage)
int ink_HTM_Scroll              (int iPageID, int iDirection)
int ink_HTM_StopScroll          (int iPageID)
int ink_HTM_SetFormValue        (int iPageID, char *pcValue)
int ink_HTM_GetFormValue        (int iPageID, char *pcValue)
int ink_HTM_CheckForPage        (char *pcUrl)
int ink_HTM_GetImgUrl           (int iPageID, int iImageIndex, char *pcUrl)
```

• Mail functions

```
int ink_MAI_Send                (MAILDATA *Mail)
int ink_MAI_Count               (int *Count)
int ink_MAI_Get                 (MAILDATA *MailArray, int MaxTo Receive,
                                 int StartMail)
int ink_MAI_Delete              (int StartWithMailNr, int EndWithMailNr)
```

• File functions

```
int ink_FTP_PutFile             (char*Hostname, char *UserID, char *Password,
                                 char *RemotePath, char *FileBuffer, short Type)
int ink_FTP_GetFile             (char*Hostname, char *UserID, char *Password,
                                 char *RemotePath, char *LocalFile, short Type)
```

• News functions

```
int ink_NWS_PostMessage         (NEWSDATA *nd)
int ink_NWS_GetMessage          (NEWSDATA *nd)
int ink_NWS_Overview            (NEWSDATA *nd)
```

• Account functions

```
int ink_AAG_CheckAccount        (void)
int ink_AAG_CreateAccount       (void)
int ink_AAG_LastUsedInfo        (void)
int ink_AAG_RequestAccount      (void)
int ink_AAG_UseStoredAccount    (void)
int ink_AAG_UpdateAccount       (void)
```

• Control & Support functions

```
int ink_CTR_Init                (INKINIT *psll)
int ink_CTR_Exit                (void)
int ink_CTR_CheckResp           (int iAnr, int *piStatus)
int ink_CTR_SetReadyFlag        (void)
int ink_CTR_GetError            (int iHandle)
int ink_CTR_GetProgress         (int *piCurrent, int *piTotal)
int ink_CTR_Cancel              (int iAnr)
int ink_CTR_ConvertToHTML       (char *pcData, int iDatatype, char *pcHtml,
                                 char *pcTitle)
intink_CTR_SetPrefString        (char *pcString, int iType)
intink_CTR_GetAllPrefString     (char **ppcString, int iType, int iWhere)
intink_CTR_DelPrefString        (char *pcString, int iType)
```

Stefan Luyten

■ UltraC and fpu crashes

Applications written in UltraC,and using floating-point arithmetic crash when accessing the fpu. The reason for this is that UltraC handles floating-point in a different way than the 3.2.3 compiler. The last one uses the “math”module that’s present in the CD-i’s ROM to execute the floating-point instructions with traps.

UltraC only uses in-line floating-point instructions. The implies that we need to emulate the fpu by software. This is possible by means of the “fpu”module - which you can install with the “p2init”utility.

Use the following code to install the floating-point emulator.

```
#include <string.h>
#include <process.h>
#include <types.h>

int fInstallFpu()
{
  process_id pid, wait_pid;
  status_code mod_status;
  error_code errnum;
  if ((errnum = _os_fork(0, 0, “p2init”,
   “fpu”, 4, 0, &pid, 0, 0)) != 0)
  {
    return SYSERR;
  }

  printf(“_os_fork() succeeded for pid
   %d\n”, pid);
  do
  {
    if((errnum = _os_wait( &wait_pid,
      &mod_status )) != 0)
    {
      printf(“_os_wait() failed:
      %02d:%02d\n”, errnum/256,
      errnum%256);
      return SYSERR;
    }
  }
  while ((wait_pid != -1) && (wait_pid !=
   pid));
  return OK;
}
```

■ How to contact PIMC

- **PIMC SUPPORT EMAIL**

  from Internet:support@pimc.be
  from CompuServe:
  INTERNET:support@pimc.be

- **MAILSERVER**

  from Internet:mailserv@pimc.bc
  from CompuServe:
  INTERNET:mailserv@pimc.be

- **PIMC SUPPORT PHONE**

  +32 11 242546

- **PIMC SUPPORT FAX**

  +32 11 242273

- **PIMC SUPPORT MAILING ADDRESS**

  Philips Interactive Media Centre
  Support
  Maastrichterstraat 63
  B-3500 Hasselt
  Belgium

- **EDITORIAL**

  from Internet:ie@pimc.be
  from CompuServe:
  INTERNET:ie@pimc.be

The Interactive Engineer
is a publication of Philips Interactive Media
Its purpose is to provide up-to-date information on CD-i technology to all supported developers and PIM software engineers.