Compañía Hasar SAIC

Manual de Drivers para Controladores Fiscales

INDICE TEMATICO

Introducción

El impresor fiscal no es un impresor común. En principio, a diferencia de la mayoría de las impresoras corrientes, no se conecta a un puerto paralelo sino a uno serie. Lleva un protocolo de comunicación empaquetado, por lo que resulta un poco complicada su implementación a nivel software. Otra característica es que la estética de la impresión queda a cargo del impresor. Este recibe solamente datos sueltos (como nombre de artículo, precio, etc.), y decide el formato de impresión. La idea de estos drivers es permitir una capa que facilite la comunicación con el impresor fiscal, a la vez que provea cierta seguridad en el funcionamiento. Existe una versión para DOS en forma de driver residente/driver de dispositivo, DLLs en 16 y 32 bits para Windows 3.x/95/NT, y en forma de demonio en UNIX SCO Open Server/System V.

DOS

Existen dos soluciones para DOS: un par de drivers, un programa stand-alone externo o una librería. En caso de decidir usar los drivers, se trata de dos programas: uno residente (LPTFIS.EXE) y un device driver opcional (FISCAL.SYS).

El Driver Residente: LPTFIS.EXE

Se encarga de redireccionar las salidas del puerto de impresión hacia el impresor fiscal (en un puerto serie). Se puede instalar directamente desde la línea de comandos de DOS. El puerto serie a usar y el paralelo a virtualizar se definen mediante parámetros (-s y -p respectivamente). Una vez instalado, se puede desinstalar con la opción -d. En caso de que el puerto no esté instalado por el BIOS, o se use una configuración no estándar, las opciones –qn y –bn ajustan el IRQ (en formato decimal) y la dirección de E/S (en hexadecimal) respectivamente.
Siempre que el programa esté residente, se puede imprimir como si de una impresora común se tratara, y el programa transmitirá exactamente cada línea al impresor. El comando se asumirá como terminado cuando se reciba un carácter **NEWLINE** (ASCII 10) o **CARRIAGE RETURN** (ASCII 13). El comando debe tener el siguiente formato:

n[parámetros]

donde:

n: carácter ASCII que corresponde al comando fiscal deseado, por ejemplo, "Y" para pedir la fecha/hora fiscal.
[parámetros]: lista de parámetros (dependiente de cada comando) que lleva el comando fiscal. Cada parámetro debe ir separado por el carácter FIELD SEPARATOR (FS, ASCII 28).

El estado del impresor será equiparado al de un impresor común, emulando los bits en la palabra de respuesta del BIOS: PRINTER BUSY, PRINTER OFFLINE y TRANSFER ERROR.

En el caso de disponer la facilidad de llamadas a interrupciones desde el lenguaje que se esté programando, existen una serie de facilidades adicionales:

La interrupción por default es BIOS PRINTER (17h), y los servicios son los siguientes:

AH: 0 **PRINT BYTE**
AL: Byte a imprimir.
DX: Número de puerto paralelo

Devuelve (en el caso que el byte enviado sea el terminador)

BX: Estado del impresor fiscal
DX: Estado del controlador fiscal
AX: Estado del impresor según BIOS.

AH: 1 **INIT PRINTER**
DX: Número de puerto paralelo.

Devuelve

BX: Estado del impresor fiscal
DX: Estado del controlador fiscal
AX: Estado del impresor según BIOS.

AH: 2 **PRINTER STATUS**
DX: Número de puerto paralelo.

Devuelve

BX: Estado del impresor fiscal
DX: Estado del controlador fiscal
AX: Estado del impresor según BIOS.

AH: 3 **UNINSTALL DRIVER**

Devuelve

AX = 0 si fue exitoso.

AH: 4 **CHECK DRIVER**

Devuelve

AH = 1234h si está corriendo.

AH: 5 **GET ANSWER**

BX:DX Dirección de un buffer donde copiar la última respuesta fiscal. Consultar el formato en la sección "El Device Driver: FISCAL.SYS".

Ejemplo de implementación en DOS:

El siguiente es un programa de ejemplo escrito en diversos lenguajes de programación DOS.

```
/* DEMOSTRACION DE DRIVER FISCAL ESCRITO EN BORLAND C */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dos.h>

void SendPaqueteFiscal (int LptNro, char *String, unsigned *FiscalStatus,
                        unsigned *PrinterStatus);
void GetStatus (int LptNro, unsigned *FiscalStatus, unsigned
                        *PrinterStatus);
int
main (void)
{
    unsigned FiscalStatus, PrinterStatus;
    int LptNro = 2;

    GetStatus (LptNro, &FiscalStatus, &PrinterStatus);
    printf ("GetStatus: Fiscal = %04x; Printer = %04x\n", FiscalStatus,
            PrinterStatus);

    SendPaqueteFiscal (LptNro, "X""\x1c""971231""\x1c""235900\n",
            &FiscalStatus, &PrinterStatus);
    printf ("SendPaquete: Fiscal = %04x; Printer = %04x\n", FiscalStatus,
            PrinterStatus);

    return 0;
}

void
GetStatus (int LptNro, unsigned *FiscalStatus, unsigned *PrinterStatus)
{
    union REGS InRegs, OutRegs;

    InRegs.x.dx = LptNro - 1;
    InRegs.h.ah = 0x02;       // Pido Status
    int86 (0x17, &InRegs, &OutRegs);

    *FiscalStatus = OutRegs.x.dx;
    *PrinterStatus = OutRegs.x.bx;
}

void
SendPaqueteFiscal (int LptNro, char *String, unsigned *FiscalStatus,
            unsigned *PrinterStatus)
{
    union REGS InRegs, OutRegs;
    char *p = String;

    InRegs.x.dx = LptNro - 1;
    InRegs.h.ah = 0x00; // PrintChar

    do {
           InRegs.h.al = *p;
           int86 (0x17, &InRegs, &OutRegs);
     } while (*(p++) != '\n');

    *FiscalStatus = OutRegs.x.dx;
    *PrinterStatus = OutRegs.x.bx;
}
```

```
Program Fiscal; { DEMOSTRACION DRIVER FISCAL ESCRITO EN TURBO PASCAL }

Uses Crt, Dos;

Const LptNro = 2;
Var PrinterStatus, FiscalStatus: Integer;

Procedure GetStatus (LptNro: Integer; Var FiscalStatus, PrinterStatus:
Integer);

   var Regs: Registers;

Begin

   Regs.dx := LptNro - 1;
   Regs.ah := $2;      { Pido Status }
   Intr ($17, Regs);

   FiscalStatus := Regs.dx;
   PrinterStatus := Regs.bx;

end;

Procedure SendPaqueteFiscal (LptNro: Integer; Cadena: String; Var
           FiscalStatus, PrinterStatus: Integer);

Var
   Regs    : Registers;
   i       : Integer;

Begin

   i := 1;
   while (i <> Length(Cadena) and Cadena[i] <> #13) do
   Begin
     Regs.dx := LptNro - 1;
     Regs.ah := $0;    { PrintChar }
     Regs.al := Ord(Cadena[i]);
     Intr ($17, Regs);
     i := i + 1;
    end;

   FiscalStatus := Regs.dx;
   PrinterStatus := Regs.bx;
end;

Begin
   GetStatus (LptNro, FiscalStatus, PrinterStatus);
   Writeln ('GetStatus: Fiscal = ', FiscalStatus, ' Printer = ',
           PrinterStatus);

   SendPaqueteFiscal (LptNro, 'X' + #28 + '971231' + #28 + '235900' +
           #10, FiscalStatus, PrinterStatus);

   Writeln ('SendPaquete: Fiscal = ', FiscalStatus, ' Printer = ',
           PrinterStatus);
end.
```

```
' PROGRAMA DEMOSTRACION DRIVER FISCAL ESCRITO EN QUICK BASIC 4.5

CONST LptNro = 2

DECLARE SUB GetStatus (LptNro As Integer, FiscalStatus As Integer,
PrinterStatus As Integer)
DECLARE SUB SendPaqueteFiscal (LptNro As Integer, Cadena As String,
FiscalStatus As Integer, PrinterStatus As Integer)

Dim PrinterStatus As Integer
Dim FiscalStatus As Integer

GetStatus LptNro, FiscalStatus, PrinterStatus
PRINT  "GetStatus: Fiscal = "; FiscalStatus; " Printer = "; PrinterStatus

SendPaqueteFiscal LptNro, "X" + Chr$(28) + "971231" + Chr$(28) + "235900"
      + Chr$(10), FiscalStatus, PrinterStatus

PRINT "SendPaquete: Fiscal = "; FiscalStatus; " Printer = ",
PrinterStatus

END

SUB GetStatus (LptNro As Integer, FiscalStatus As Integer, PrinterStatus
As Integer)

    Dim Regs As RegType

    Regs.dx = LptNro - 1
    Regs.ax = &H0200          ' Pido Status (AH = 2)
    CALL INTERRUPT (&H17, Regs, Regs)

    FiscalStatus = Regs.dx
    PrinterStatus = Regs.bx

END SUB

SUB SendPaqueteFiscal (LptNro As Integer, Cadena As String, FiscalStatus
            As Integer, PrinterStatus As Integer)

Dim Regs As RegType
Dim i As Integer

    Regs.dx = LptNro - 1
    Regs.ax = &H0       ' PrintChar AH=0

    i = 1
    while (i <> Length(Cadena) and Mid$(Cadena,i,1) <> Chr$(13))
      Regs.ax = Mid$(Cadena,i,1)'El byte bajo queda con el carácter
                                'y el más alto queda en cero (Print Byte)
      CALL INTERRUPT (&H17, Regs, Regs)
      i = i + 1
    wend

    FiscalStatus = Regs.dx
    PrinterStatus = Regs.bx

END SUB
```

El programa pide el estado del impresor, para después setear la hora y la fecha. El carácter **`\x1c, $28 o Chr$(28)`** es el separador de campos (ASCII 28). Se deberá buscar en la documentación del lenguaje sobre el que se desarrolla la forma de acceder a las interrupciones.

Funcionamiento bajo Windows 3.x/9x:

Cuando se quiere usar el driver bajo ambiente Windows, conviene "mapear" un puerto de impresión no usado por el sistema, es decir, sobre el cual no haya ninguna impresora instalada. Digamos, si existe sobre LPT1: una impresora (aunque ésta físicamente no esté presente), el driver hay que instalarlo con la opción **-p2**, para indicar LPT2 como puerto a virtualizar. Si bajo Windows 9x existen problemas de comunicaciones, conviene cargar el TSR después de haber cargado Windows, esto es, desde la ventana de DOS donde va a ser usado. En Win98SE si existen problemas, habrá que correr un programa cada vez que se inicia la máquina (carpeta Inicio) llamado W98SECOM.EXE que corrige un bug de éste.

Consideraciones de implementación:

Al tener un puerto redireccionado, es perfectamente plausible abrir el puerto como si de un archivo se tratara, y escribir en él. Claro que no se posee "feedback" de parte del controlador fiscal, pero un código como el que sigue es correcto si bien ineficiente:

```
OPEN "LPT1" FOR BINARY AS #1

'Mando un comando (X Fiscal)
Comando = "9" + CHR$(28) + "X" + CHR$(10)
PUT #1, , Comando
```

Donde "LPT1" es el puerto redireccionado con la opción -p en el residente. También puede ser "PRN" si el puerto es LPT1. La siguiente sentencia desde la línea de comandos de DOS es factible también:

```
C:\>ECHO 9^\X> LPT1
```

o bien:

```
C:\>TYPE XFISCAL.TXT > PRN
```

donde XFISCAL.TXT es un archivo de texto ASCII con el comando.
Este tipo de comandos trabajan pidiendo un status por cada carácter que mandan, status que se traduce a un pedido de status al impresor fiscal. Esto hace lenta la operación, por lo que el uso de estos comandos de DOS son sólo útiles en el campo de la experimentación.

Detección de problemas:

En el paquete con los drivers viene incluído un driver adicional para permitir una primera aproximación a la implementación. Este driver es en todo idéntico al anterior (LPTFIS.EXE), pero muestra en la pantalla todos los detalles de la comunicación, pudiendo ver el desarrollo del protocolo.
Si se le especifica una variable de ambiente llamada DEBUGLEVEL con un número de uno a tres, se puede filtrar la información. A más bajo el número, más reducida es la información mostrada. Si esta variable no se especifica, se muestra toda la información disponible.

Otras Opciones:

Se puede especificar –v, -b y –q para cambiar respectivamente velocidad, I/O base e IRQ del puerto. También se puede usar el protocolo con STAT_PRN (consultar apéndice) usando la opción –n. La opción –a agrega un retorno de carro al final de la respuesta junto con la marca de fin de archivo.

El Device Driver: FISCAL.SYS.

Este programa es un device driver instalable que facilita un poco la interface con el driver residente (LPTFIS.EXE) y depende de éste para funcionar. Para instalarlo, hay que agregar la siguiente línea en el archivo config.sys:

DEVICE=FISCAL.SYS n

o bien:

DEVICEHIGH=FISCAL.SYS n

si se quiere cargar en memoria alta. Este driver lleva un parámetro obligatorio (n) que indica qué puerto paralelo virtualizará LPTFIS.EXE. Hay que tener cuidado de que ambos drivers estén "sincronizados".
Una vez instalado, existirá un archivo especial de sistema llamado **FISPRN**, que se podrá abrir como un archivo común y corriente, y si se escribe en él, se escribirá en el controlador fiscal, y si se lee de él, se leerán las respuestas fiscales. Estas respuestas fiscales asumirán la siguiente forma:

`PrinterStatus<FS>FiscalStatus<FS>Campos…`

donde:

FS:	Field Separator, carácter ASCII 28.
PrinterStatus:	Status del impresor, 4 bytes en formato hexadecimal
FiscalStatus:	Status fiscal, 4 bytes en formato hexadecimal
Campos…:	Campos de la respuesta fiscal, separados por el carácter ASCII 28. Su longitud depende de cada comando.

En el caso de estar usando la opción –n en lptfis.exe, la respuesta especial STAT_PRN podrá diferenciarse por tener como formato `PrinterStatus<FS>FiscalStatus<FS>~STATPRN~`. Consultar el apéndice para constatar el funcionamiento de STAT_PRN.

Conflictos de interrupciones (Entornos de Red):

Bajo ciertos ambientes pueden existir conflictos de interrupciones, por ejemplo, bajo Novell. Estos conflictos se deben a que el sistema operativo (u otros programas) usan la interrupción de impresión para una tarea excluyente. Ante esta situación, se puede cambiar la interrupción al driver de dispositivo y al driver residente, mediante un parámetro adicional que indica el número de interrupción expresado en formato hexadecimal. En el driver de dispositivo, es el segundo parámetro, por ejemplo:

```
DEVICE = fiscal.sys 2 90
```

donde 2 sigue siendo el número de LPT, pero la interrupción usada será la 90 hexadecimal, en vez de la 17.
En el TSR, el argumento "`-ix`", (donde x es un número hexadecimal) indica el número de interrupción. Por ejemplo:

```
lptfis -p2 -s3 -i90
```

donde `-p2` indica el número de LPT, `-s3` el número de COM, y `90` la interrupción a usar en vez de la 17.

Implementación:

Su implementación es muy simple. Sólo hay abrir el archivo, y leer/escribir como si de cualquier archivo se tratara.

/ PROGRAMA DEMOSTRACION DEL DEVICE DRIVER FISCAL ESCRITO EN BORLAND C */*

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stdlib.h>

#define DRIVER "fisprn"
#define MAX_BUFFER 500

char Buffer[MAX_BUFFER+1];

int
main (void)
{
      int fd;

      /* Abro el driver */
      if ((fd = open(DRIVER, O_RDWR | O_BINARY )) < 0)
      {
            printf ("No pude abrir %s\n", DRIVER);
            exit (1);
      }

      /* Mando un comando (X Fiscal) */
      if (write (fd, "9""\x1c""X\n", 4) < 0)
      {
            printf ("Write Fault\n");
            exit (1);
      }

      /* Leo la respuesta */
      if (read (fd, Buffer, MAX_BUFFER) < 0)
      {
            printf ("Read Fault\n");
            exit (1);
      }

      /* Muestro la respuesta */
      printf ("Respuesta: %s\n", Buffer);

      return 0;
}
```

```
'Programa de demostración del device driver fiscal
'escrito en MS-DOS QBasic Versión 1.1 (DOS 6.22)

        DIM Comando AS STRING
        DIM Respuesta AS STRING * 500

        'Abro el driver
        ON ERROR GOTO ErrorOpen
        OPEN "fisprn" FOR BINARY AS #1

        'Mando el comando (X Fiscal)
        ON ERROR GOTO ErrorPut
        Comando = "9" + CHR$(28) + "X" + CHR$(10)
        PUT #1, , Comando

        'Leo la respuesta
        ON ERROR GOTO ErrorGet
        GET #1, , Respuesta

        'Muestro la respuesta
        PRINT "Respuesta: "; Respuesta
        END
ErrorOpen:
        PRINT "Error abriendo driver"
        END
ErrorPut:
        PRINT "Error escribiendo comando"
        END
ErrorGet:
        PRINT "Error obteniendo respuesta"
        END
```

```
{ Demostración del device driver fiscal escrito en Turbo Pascal }

Program Prueba;

uses Crt;

Const
      Driver = 'FISPRN';

Var
      fd          : File;
      Comando     : String[10];
      Buffer      : Array [0..500] of Char;
      Result      : Integer;

Begin
      { Abro el driver }
      {$I-}
      Assign (fd, Driver);
      Reset (fd,1);
      {$I+}
      if (IOResult <> 0) then
      Begin
            Writeln ('Error abriendo ', Driver);
            halt(1);
      end;

      { Mando el comando: X Fiscal }
      Comando := '9' + #28 + 'X' + #10;
      BlockWrite (fd, (@Comando[1])^, Length(Comando), Result);
      if (Result < 0) then
      Begin
            Writeln ('Error en comando');
            halt (1);
      end;

      { Leo la respuesta }
      BlockRead (fd, Buffer, sizeof(Buffer), Result);
      if (Result < 0) then
      Begin
            Writeln ('Error en respuesta');
            halt(1);
      end;

      { Muestro la respuesta }
      Writeln ('Respuesta: ', Buffer);
      close (fd);
end.
```

```
// Demostración de uso del device driver fiscal escrito en Clipper

#define DRIVER "fisprn"
#define MAXBUF 500

local Respuesta := space(MAXBUF)

* Abro el driver:
fd = fopen(DRIVER,2)
if fd < 0
      ? "Error abriendo el archivo", DRIVER, "; Dos error" ,
            ltrim(str(Ferror())), chr(10)
      quit
endif

* Escribo el comando:
Comando = "9" + chr(28) + "X" + chr(10)
if fwrite (fd, Comando, len(Comando)) < 0
      ? "Error escribiendo, Dos error", ltrim(str(Ferror())), chr(10)
      quit
endif

* Leo la respuesta
if fread (fd, @Respuesta, MAXBUF) < 0
      ? "Error en respuesta; Dos error", ltrim(str(Ferror())), chr(10)
      quit
endif

* Imprimo la respuesta:
? "Respuesta:", Respuesta
fclose(fd)
```

```
IDENTIFICATION DIVISION.
* DEMOSTRACION DEL USO DEL DRIVER FISCAL EN RMCOBOL85 version 6.10
PROGRAM-ID.
AUTHOR.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM.
OBJECT-COMPUTER. IBM.
SPECIAL-NAMES. DECIMAL-POINT IS COMMA.
*******************************
INPUT-OUTPUT SECTION.
FILE-CONTROL.
*******************************
    SELECT FISPRN ASSIGN "FISPRN"
    ORGANIZATION IS line SEQUENTIAL
    FILE STATUS IS FIL-STATUS.
    SELECT FISPRN2 ASSIGN "FISPRN"
    ORGANIZATION IS binary SEQUENTIAL
    FILE STATUS IS FIL-STATUS2.
*******************************
DATA DIVISION.
FILE SECTION.
*******************************
FD  FISPRN.
01  R-FISPRN.
    03  FILLER          PIC X(500).
FD  FISPRN2.
01  R-FISPRN2.
    03  printer-status  PIC x(4).
    03  filler          PIC x.
    03  fiscal-status   PIC x(4).
    03  filler          PIC x.
    03  nro-ticket      PIC 9(8).
    03  FILLER          PIC X(482).
*******************************
WORKING-STORAGE SECTION.
*******************************
* Comandos para poder imprimir un ticket
*******************************
01  l-abro-ticket.
    03  FILLER          PIC X           VALUE "@".
    03  FILLER          PIC X           VALUE H"1C".
    03  FILLER          PIC X           VALUE "T".
    03  FILLER          PIC X           VALUE H"1C".
    03  FILLER          PIC X           VALUE "T".
01  l-item.
    03  FILLER          PIC X           VALUE "B".
    03  FILLER          PIC X           VALUE H"1C".
    03  DESCRI          PIC X(20)       VALUE SPACES.
    03  FILLER          PIC X           VALUE H"1C".
    03  CANTI           PIC 999,99      VALUE ZEROS.
    03  OTROS-DECI      PIC X(8)        VALUE "00000000".
    03  FILLER          PIC X           VALUE H"1C".
    03  SIGNO-MONTO     PIC X           VALUE SPACES.
    03  MONTO           PIC 9(6),99     VALUE ZEROS.
    03  FILLER          PIC X           VALUE H"1C".
    03  PORCENIVA       PIC 99,99       VALUE ZEROS.
    03  OTROS-DECI-INTER PIC X(6)       VALUE "000000".
    03  FILLER          PIC X           VALUE H"1C".
    03  CUALQUIERA      PIC 9           VALUE 0.
```

```
    03  FILLER          PIC X           VALUE H"1C".
    03  CALIFI-MONTO    PIC X           VALUE "T".
01  l-total.
    03  FILLER          PIC X           VALUE "D".
    03  FILLER          PIC X           VALUE H"1C".
    03  FILLER          PIC X(30)       VALUE "TOTAL DEL TICKET".
    03  FILLER          PIC X           VALUE H"1C".
    03  MONTO-PAGO      PIC 9(9),99     VALUE ZEROS.
    03  FILLER          PIC X           VALUE H"1C".
    03  TIPO-OPE        PIC X           VALUE "T".
    03  FILLER          PIC X           VALUE H"1C".
    03  CUALQUIERA      PIC X           VALUE 0.
01  l-cerrar.
    03  FILLER          PIC X           VALUE "E".
01  l-controlo.
    03  FILLER          PIC X           "*".
01  FIL-STATUS.
    03  STS-1           PIC X           VALUE SPACES.
    03  STS-2           PIC X           VALUE SPACES.
01  FIL-STATUS2.
    03  STS2-1          PIC X           VALUE SPACES.
    03  STS2-2          PIC X           VALUE SPACES

PROCEDURE DIVISION.
DECLARATIVES.
E-FISPRN SECTION 0.
    USE AFTER STANDARD ERROR PROCEDURE ON FISPRN.
S-FISPRN.
END DECLARATIVES.
TRONCO SECTION.
OPEN OUTPUT FISPRN
    WRITE R-FISPRN FROM l-controlo
    CLOSE FISPRN
    OPEN INPUT FISPRN2
    READ FISPRN2
OPEN OUTPUT FISPRN
    WRITE R-FISPRN FROM l-abro-ticket
    CLOSE FISPRN
    PERFORM IMPRI-VARIOS THRU F-IMPRI-VARIOS
    PERFORM IMPRI-ITEM THRU F-IMPRI-ITEM VARYING
            INDIC2 FROM 1 BY 1 UNTIL INDIC2 > CANT-IT.
OPEN OUTPUT FISPRN
    WRITE R-FISPRN FROM l-cerrar
    CLOSE FISPRN
    OPEN INPUT FISPRN2
    READ FISPRN2
CLOSE FISPRN2
F-IMPRIMIR-FACTURAB.    EXIT.
*
IMPRI-ITEM.
    OPEN OUTPUT FISPRN
    PERFORM ARMO-LINEA THRU F-ARMO-LINEA
    WRITE R-FISPRN FROM L-ITEM
    CLOSE FISPRN
F-ARMO-LINEA.   EXIT.
```

```
*
* Este programa realiza un ticket y una factura A
* para FoxPro DOS 2.0
*

      fp = fopen("fisprn",2)

      ? 'fopen devolvio ' + str(fp)

      if fp < 0
            Wait Window "ERROR: NO SE PUEDE USAR EL CONTROLADOR FISCAL"
            quit
      Endif

      Se=CHR(28)
      STORE SPACE(512) TO Respuesta
      SET CONSOLE ON

      * Inicializo el Printer
      =InitFiscal ()

      * Cancela un ticket que haya quedado por la mitad
      =CancelTicket ()

      * Cierra el ticket que no pudo ser cerrado
      =CloseTicket ()

      *
      * Establece el texto de encabezamiento y el texto de cola del
      * ticket
      *

      =SetHeaderTrailer ()

      * Imprime el ticket
      =ImprimirTicket ()

    * Imprime una factura
    =ImprimirFactura ()

      =fclose (fp)

* ***
* Sincroniza la numeracion de los paquetes que envia el driver
* con aquella que lleva el impresor fiscal
* ***

function InitFiscal

      s = "*" + CHR(10)
      =Enviar (s)

      s = "*" + CHR(10)
      =Enviar (s)

return
```

```
* ***
* Cancela el ticket
* ***

function CancelTicket

      s = "D" + Se + " " + Se + "0.00" + Se + "C" + Se + "0" + CHR(10)
      =Enviar (s)

return

* ***
* Cierra el ticket
* ***

function CloseTicket

      s = "E" + CHR(10)
      =Enviar (s)

return

* ****
* ** Funcion: SetHeaderTrailer
* ****

FUNCTION SetHeaderTrailer

*
* Establece las lineas a imprimir en el header y en el trailer.
* Esto NO hace falta hacerlo para cada ticket.
*

for i=1 to 10

      linea = alltrim(str(i))
      s = "]" + Se + linea + Se + "Linea " + linea + ;
            " de Header" + CHR(10)

      * Borra la línea de header
      * s = "]" + Se + linea + Se + CHR(127) + CHR(10)

      =Enviar (s)

next

for i=11 to 20

      linea = alltrim(str(i))
      s = "]" + Se + linea + Se + "Linea " + linea + ;
            " de Trailer" + CHR(10)

      * Borra la línea de trailer
      * s = "]" + Se + linea + Se + CHR(127) + CHR(10)

      =Enviar (s)

next
RETURN 0
```

```
* ***
* Imprime un ticket completo
* ***

function ImprimirTicket

      * Abre el ticket
      s = "@" + Se + "T" + Se + "T" + CHR(10)
      =Enviar(s)

      * Imprime un texto no fiscal
      s = "A" + Se + "Texto Fiscal" + Se + "0" + CHR(10)
      =Enviar (s)

      * Vende un producto
      s = "B" + Se + "Plu Nro 1" + Se + "5.000" + Se + "100.00" + Se + ;
            "18.00" +   Se + "M" + Se + "0.00" + Se + "0" + Se ;
            + "T"  + CHR(10)
      =Enviar (s)

      * Realiza el pago
      s = "D" + Se + "Pago Nro 1" + Se + "2222.00" + Se + "T" + Se + ;
            "0" + CHR(10)
      =Enviar (s)

      * Cierra el ticket
      s = "E"     + CHR(10)
      =Enviar (s)

return

 * ****
* ** Funcion: ImprimirFactura
* ****

FUNCTION ImprimirFactura

*
* Datos del comprador (Responsable Inscripto)
*

s = "b" + Se + "Juan Perez" + Se + "20183697308" + Se + "I" + Se + ;
      "C" + CHR(10)
=Enviar (s)

*
* Abre un comprobante fiscal de tipo factura A
*

s = "@" + Se + "A" + Se + "T" + CHR(10)
=Enviar (s)
```

```
*
* Venta de los articulos
*

* Descripcion adicional como texto fiscal
s = "A" + Se + "Descripcion adicional ... " + Se + "0" + CHR(10)
=Enviar (s)

s = "B" + Se + "Articulo Nro 1" + Se + "5.000" + Se + "100.00" + Se ;
      + "18.00" + Se + "M" + Se + "0.00" + Se + "0" + Se + "B" + CHR(10)
=Enviar (s)

s = "B" + Se + "Articulo Nro 2" + Se + "4.000" + Se + "50.00" ;
      + Se + "21.00" + Se + "M" + Se + "0.00" + Se + "0" ;
      + Se + "B" + CHR(10)
=Enviar (s)

*
* Percepciones a aplicar
*

s = "`" + Se + "*.*" + Se + "Percepcion A" + Se + "20.00" + CHR(10)
=Enviar (s)

s = "`" + Se + "*.*" + Se + "Percepcion B" + Se + "15.00" + CHR(10)
=Enviar (s)

*
* Realiza el pago
*

s = "D" + Se + "Pago Nro 1" + Se + "2222.00" + Se + "T" + Se + ;
      "0" + CHR(10)
=Enviar (s)

*
* Cierra el ticket
*

s = "E" + CHR(10)
=Enviar (s)

RETURN 0
```

```
*****
* Funcion: Enviar
*
* Envia un comando al impresor fiscal y analiza la respuesta.
*****

FUNCTION Enviar

PARAMETERS String

PRIVATE Result

? "Comm: " + String

* Si la funcion fwrite retorna un numero menor que cero, retorna.
* Esto puede ser por un problema de comunicaciones con el impresor.

n = fwrite (fp, String)

if n = 0
      ? "Error enviando el comando"
      RETURN -1
ENDIF

* Levanta la respuesta.
Respuesta = fread (fp, 512)

* Analiza si existe algun error.
if GetErrors (Respuesta) < 0
      return -1
endif

? " "

RETURN 0
```

```
****
* FUNCTION GetErrors
*
* Esta funcion levanta la respuesta del printer e imprime en
* el mensaje de error si es que existe.
****

FUNCTION GetErrors

PARAMETERS Resp

PRIVATE Origen, OffsetSep, i, c

DECLARE FiscalErrors [16]
DECLARE PrinterErrors[16]

FiscalErrors[1] =       "Error en chequeo de memoria fiscal"
FiscalErrors[2] =       "Error en chequeo de la memoria de trabajo"
FiscalErrors[3] =       "Carga de bateria baja"
FiscalErrors[4] =       "Comando desconocido"
FiscalErrors[5] =       "Datos no validos en un campo"
FiscalErrors[6] =       "Comando no valido para el estado fiscal actual"
FiscalErrors[7] =       "Desborde del total"
FiscalErrors[8] =       "Memoria fiscal llena"
FiscalErrors[9] =       "Memoria fiscal a punto de llenarse"
FiscalErrors[10] =      ""
FiscalErrors[11] =      ""
FiscalErrors[12] =      "Error en ingreso de fecha"
FiscalErrors[13] =      "Recibo fiscal abierto"
FiscalErrors[14] =      "Recibo abierto"
FiscalErrors[15] =      "Factura abierta"
FiscalErrors[16] =      ""

PrinterErrors[1]  = ""
PrinterErrors[2]  = ""
PrinterErrors[3]  = "Error de Impresora"
PrinterErrors[4]  = "Impresora Offline"
PrinterErrors[5]  = "Falta papel del diario"
PrinterErrors[6]  = "Falta papel de tickets"
PrinterErrors[7]  = "Buffer de Impresora lleno"
PrinterErrors[8]  = ""
PrinterErrors[9]  = ""
PrinterErrors[10] = ""
PrinterErrors[11] = ""
PrinterErrors[12] = ""
PrinterErrors[13] = ""
PrinterErrors[14] = ""
PrinterErrors[15] = ""
PrinterErrors[16] = ""

Origen = 1

OffsetSep = AT ( CHR(28), Resp )
```

```
* Convierte en entero el status del impresor
PrinterStatus = HexaToInt (SUBSTR ( Resp, Origen, OffsetSep - 1))

IF PrinterStatus < 0
      RETURN -1
ENDIF

Origen = OffsetSep

* Analiza los bits comenzando del menos significativo
FOR i = 1 TO 16
      IF ( INT (PrinterStatus % 2) == 1 )
            IF ( LEN (PrinterErrors[i]) > 0 )
                  ? "PrinterStatus: " + PrinterErrors[i]
            ENDIF
      ENDIF
      PrinterStatus = PrinterStatus / 2
NEXT

OffsetSep = AT ( CHR(28), SUBSTR (Resp, Origen + 1) )

IF OffsetSep == 0
      OffsetSep = LEN(Resp)
ENDIF

* Convierte en entero el status fiscal
FiscalStatus = HexaToInt (SUBSTR (Resp, Origen + 1, OffsetSep - 1))

IF FiscalStatus < 0
      RETURN -1
ENDIF

* Analiza los bits comenzando del menos significativo
FOR i = 1 TO 16
      IF ( INT (FiscalStatus % 2) == 1 )
            IF ( LEN (FiscalErrors[i]) > 0 )
                  ? "FiscalStatus: " + FiscalErrors[i]
            ENDIF
      ENDIF
      FiscalStatus = FiscalStatus / 2
NEXT

RETURN 0
```

```
****
* FUNCTION HexaToInt
*
* Esta funcion convierte un numero hexadecimal en su equivalente
* en binario.
****

FUNCTION HexaToInt

PARAMETERS HexValue

PRIVATE i, Value, Status

? 'HexValue = ' + HexValue

Status = 0

FOR i = 4 TO 1 STEP -1

      S = SUBSTR(HexValue, i, 1)

      Value = ASC (S)

      IF ( Value >= ASC('A') AND Value <= ASC('F') )

            Value = Value - ASC('A') + 10
            Status = Status + Value * (16 ** ( 4 - i ))
            next

      ENDIF

      IF ( Value >= ASC('a') AND Value <= ASC('f') )

            Value = Value - ASC('a') + 10
            Status = Status + Value * (16 ** ( 4 - i ))
            next

      ENDIF

      IF ( Value >= ASC('0') AND Value <= ASC('9') )

            Value = Value - ASC('0')
            Status = Status + Value * (16 ** ( 4 - i ))
            next

      ENDIF

      ? "HexaToInt: Numero hexadecimal incorrecto: " + HexValue
      RETURN -1

NEXT

RETURN Status
```

El Driver Fiscal y Clarion

Para usar el driver fiscal desde Clarion (DOS) se necesita compilar el programa junto con un módulo binario (getans.bin). Para lograr esto, en la aplicación hay que crear un procedimiento OTHERS llamado GETANS con las siguientes características:

```
Procedure Name : GETANS
Descripcion    : Módulo binario para obtener la respuesta.
Module Name    : GETANS
Binary         : Yes
Return Value   : Yes
Date Type      : Long
```

Un ejemplo del uso del driver usando el archivo binario sería:

```
REPORTE_X    PROCEDURE

ARCHIVO      DOS,ASCII,NAME('FISPRN'),PRE(TTT)
RECORD         RECORD
REGISTRO         STRING(255)
             . .
VAR           LONG
RESPUESTA     STRING(255)


CODE

OPEN(ARCHIVO)                                  ! Abro el driver
TTT:REGISTRO = '9' & CHR(28) & 'X' & CHR(10) ! Defino el comando a enviar

ADD(ARCHIVO)                                   ! Mando el comando
VAR = GETANS(RESPUESTA,23) ! Recibo la Respuesta en la variable RESPUESTA

ACTION = 0

RETURN
```

Otros Lenguajes:

En el caso de RM Cobol 5.x es posible usar el programa mediante un par de módulos auxiliares (PUTCMD.EXE y GETANS.EXE). En este caso no es necesario usar el device driver. Existen otros lenguajes donde ya no es posible usar estos drivers, como el caso de otras versiones de Cobol o lenguajes donde no sea una posibilidad abrir un archivo en modo binario. Para estos lenguajes se puede recurrir a un programa ejecutable que toma y responde comandos desde y hacia archivos ASCII reales. Este programa se llama SPOOLER.EXE y ademas tiene la capacidad de trabajar como un verdadero spooler en un entorno de red. Consultar la documentación de este programa para más información.

La librería para lenguaje C

Existe una librería para comunicarse al impresor fiscal que puede linkearse con programas escritos en lenguaje C. Esta librería está disponible en los modelos de memoria SMALL, COMPACT, MEDIUM, LARGE y HUGE. Está compilada para Borland C++ 3.1 o superior. Deberá usarse la librería que corresponda según el modelo usado para el programa.
Hay tres puntos de entrada de esta librería, a saber:

- `int OpenCommFiscal (int PortNumber);`

Abre el puerto físico. Si el puerto no se puede abrir, por alguna razón, devuelve –1. Si todo estuvo bien, devuelve un descriptor que se usará para los demás puntos de entrada.

- ```
  int MandaPaqueteFiscal (int PortDescriptor, char *Command,
                          unsigned short *FiscalStatus,
                          unsigned short *PrinterStatus,
                          char *AnswerBuffer);
  ```

Manda un paquete al puerto de comunicaciones. `PortDescriptor` es el descriptor devuelto por `OpenCommFiscal`. El parámetro `Command` es el comando a enviar, que debe tener el siguiente formato:

```
n[parámetros]
```

donde:

`n`: carácter ASCII que corresponde al comando fiscal deseado, por ejemplo, "Y" para pedir la fecha/hora fiscal.
`[parámetros]`**:** lista de parámetros (dependiente de cada comando) que lleva el comando fiscal. Cada parámetro debe ir separado por el separador de campos (ASCII 28).

`AnswerBuffer` es un puntero a un buffer con capacidad suficiente para albergar una respuesta fiscal con el siguiente formato:

```
PrinterStatus<FS>FiscalStatus<FS>Campos...
```

donde:

PrinterStatus	**:**	Status del impresor, 4 bytes en formato hexadecimal
FiscalStatus	**:**	Status fiscal, 4 bytes en formato hexadecimal
<FS>	**:**	Separador de campos (ASCII 28)
Campos	**:**	Campos de la respuesta fiscal, separados por el separador de campos (ASCII 28). Su longitud depende de cada comando**.**

Un buffer de 500 bytes es suficiente para la respuesta más larga. Los punteros `FiscalStatus` y `PrinterStatus` deben apuntar a dos variables de tipo `unsigned short` para guardar los dos estados del controlador fiscal. Esta función devolverá < 0 si algo ha ido mal, y 0 si todo estuvo bien. Consultar la tabla de errores en la sección Windows. En caso que devuelva < 0, los datos devueltos en los punteros pueden contener cualquier cosa.

- `int CloseCommFiscal (int PortDescriptor);`

Cierra las comunicaciones. El parámetro `PortDescriptor` es el devuelto por `OpenCommFiscal`.
```

- `int SetNewProtocol (int Value)`

Utiliza el nuevo protocolo con la modalidad de comandos STAT_PRN. Consultar el apéndice para más detalles.

- `int SetBaudios(int PortNumber, long Bauds)`

Cambia la velocidad del puerto si el modelo de controlador admite otra velocidad que 9600 baudios.

- `int SetKeepAliveHandler(PFV Handler)`

Si no se está usando el protocolo nuevo, en ciertos momentos no se tendrá el control durante errores de falta de papel, errores mecánicos, etc. hasta que sean arreglados en forma externa por el usuario. Para evitar este problema, se puede definir una función que será llamada cuando el impresor está ocupado o cuando halla falta de papel. La función debe tener las siguientes caracterísiticas:

- `void KeepAliveHandler(int Reason, int Port)`

En el primer parámetro se recibirá el caracter DC2 (18) cuando el impresor se reporte como ocupado o DC4 (20) en caso de falta de papel. Se recomienda devolver el control inmediatamente, a riesgo de entorpecer el desarrollo del protocolo.

- `int SearchPrn(int Handler, long *Baud)`

Realiza la busqueda del Controlador Fiscal en el Puerto asociado a Handler, en las siguientes velocidades: 1200, 2400, 4800, 9600, 19200, 38400 y 57600 baudios.
Devuelve 0 y la velocidad a la que se sincronizaron en *Baud, o un código de error (< 0) si el Controlador Fiscal no fue encontrado.

- `int SetCommandRetries (int Retries)`

Configura el número de reintentos máximo de transmisiones de paquetes fiscales transmitidos por la función 'MandaPaqueteFiscal' cuando el Controlador Fiscal no responde (Time Out).
Devuelve el número de reintentos previo a la re-configuración.

Implementación

El siguiente es un programa de ejemplo de como usar esta librería:

```
/*
      Programa de demostración de uso de librería para 'C'
*/

#include <stdio.h>
#include <stdlib.h>

#include "fislib.h"

void
SendCommandString (int Port, char *Command)
{
      static char Respuesta[500];
      unsigned short FiscalStatus, PrinterStatus;

      if (MandaPaqueteFiscal (Port, Command,
                              &FiscalStatus, &PrinterStatus,
                              Respuesta) < 0)
      {
            printf ("ERROR: Error mandando paquete\n");
            exit (1);
      }

      printf ("Respuesta: %s\n", Respuesta);
}

int
main(int argc, char *argv[])
{
      int PortNumber = atoi(argv[1]);
      int Port;

      if (argc != 2)
      {
            printf ("Uso: fiscal n\n");
            exit (1);
      }

      if ((Port = OpenCommFiscal(PortNumber)) < 0)
      {
            printf ("ERROR: No pude abrir COM%d\n", PortNumber);
            exit (1);
      }

      printf ("Corriendo sobre COM%d...\n", PortNumber);

      SendCommandString (Port, "*");" // Pido un status.

      return 0;
}
```

La librería viene con una librería adicional para el tratamiento de información de debuggeo. Esta librería se llama "debug.lib", y contiene una rutina de impresión en pantalla. Se deberá proporcionar una función análoga si se quiere usar la información de debugging para otro propósito (por ejemplo, para guardarla en un archivo), o para que simplemente no haga nada. Esta función se llama `FISdebug()`, y para implementarla deberá hacerse con el siguiente prototipo:

```
void FISdebug (int level, char *fmt, ... );
```

El parámetro `level` es un número de uno a tres que proporciona la importancia de la información. A más pequeño el número, más importancia tiene. El parámetro `fmt` es el formato del string, y a continuación vienen los argumentos opciones. El siguiente es un ejemplo de implementación de FISdebug:

```
#include <stdio.h>
#include <stdarg.h>
#include <conio.h>

extern int DesiredLevel;

void
FISdebug (int level, char *fmt, ... )
{
      va_list argptr;
      static char buffer[200];
      FILE *fp;

      /*
            Controlo el nivel de debugging.
      */

      if (level > DesiredLevel)
            return;

      va_start(argptr, fmt);
      vsprintf (buffer, fmt, argptr);

      /*
            Rutina de debugging: para este ejemplo,
            guardamos la información en un archivo.
      */

      if ( !(fp = fopen ("Fiscal.Log", "a+")) )
            return;
      fprintf(fp, "%s\n", buffer);
      fclose (fp);

      va_end (argptr);
}
```

Otros compiladores:

Para compilar los fuentes en otros compiladores, la única salvedad a tener en cuenta es que hay que definir dos MACROS: DOS y opcionalmente DEBUG para debugging. Compiladores como Microsoft C tienen una opción para deshabilitar el chequeo de desborde de pila: usarla en el caso que exista (`-Gs` en el compilador mencionado).

La librería para lenguaje Clipper

Existe también una librería para linkear con programas escritos en Clipper, evitando el uso del programa residente y el driver. Los puntos de entrada a la librería son:

- `OpenPort(PortNumber)`

Abre el port de comunicaciones determinado por PortNumber. Este indica el numero de COM al que esta conectado el printer. Devuelve un handler que debe ser usado en el resto de las funciones.

- `ClosePort(Handler)`

Cierra el port determinado por Handler.

- `InitFiscal(Handler)`

Envía dos pedidos de status al impresor para asegurarse que no haya error por número de paquete repetido.

- `MandaPaq(Handler, Comando)`

Envía un comando al impresor fiscal, con el siguiente formato:

`n[parámetros]`

donde:

`n`: carácter ASCII que corresponde al comando fiscal deseado, por ejemplo, "Y" para pedir la fecha/hora fiscal.
`[parámetros]`**:** lista de parámetros (dependiente de cada comando) que lleva el comando fiscal. Cada parámetro debe ir separado por el separador de campos (ASCII 28).

- `Respuesta(Handler)`

Lee la última respuesta del impresor fiscal. La respuesta tendrá el siguiente formato:

`PrinterStatus<FS>FiscalStatus<FS>Campos...`

donde:

PrinterStatus	**:**	Status del impresor, 4 bytes en formato hexadecimal
FiscalStatus	**:**	Status fiscal, 4 bytes en formato hexadecimal
<FS>	**:**	Separador de campos (ASCII 28)
Campos	**:**	Campos de la respuesta fiscal, separados por el separador de campos (ASCII 28). Su longitud depende de cada comando**.**

Para linkear con un programa Clipper se usa el utilitario RTLink o PLink86, por ejemplo:

```
RTLink @prueba.lnk
```

donde `prueba.lnk` contiene instrucciones válidas de linkeo para el linkeador:

```
output prueba.exe
fi prueba
lib clipper.lib, extend.lib, terminal.lib, fiscal.lib
;
```

```
*
* Este programa realiza un ticket y una factura A
* para Clipper usando la librería.
*

PARAMETERS Com

PUBLIC Handler, Port, n, Se

      * Separador de campos
      Se = CHR(28)

      n = VAL(Com)

      if ( n < 0 .OR. n > 4 )
            return
      endif

      * Abro el Port de Comunicaciones
      Handler = OpenPort (Com)

      * Inicializo el Printer
      InitFiscal (Handler)

      * Cancela un ticket que haya quedado por la mitad
      CancelTicket ()

      * Cierra el ticket que no pudo ser cerrado
      CloseTicket ()

      *
      * Establece el texto de encabezamiento y el texto de cola del
      * ticket
      *

      SetHeaderTrailer ()

      * Imprime el ticket
      ImprimirTicket ()

    * Imprime una factura
    ImprimirFactura ()

      * Cierra el Port
      ClosePort (Handler)

* ****
* ** Funcion: CancelTicket
* ****

FUNCTION CancelTicket

      s = "D" + Se + " " + Se + "0.00" + Se + "C" + Se + "0"
      Enviar (s)

RETURN 0
```

```
* ****
* ** Funcion: CloseTicket
* ****

FUNCTION CloseTicket

      s = "E"
      Enviar (s)

RETURN 0


* ****
* ** Funcion: SetHeaderTrailer
* ****

FUNCTION SetHeaderTrailer

*
* Establece las lineas a imprimir en el header y en el trailer.
* Esto NO hace falta hacerlo para cada ticket.
*

for i=1 to 10

      linea = alltrim(str(i))

      s = "]" + Se + linea + Se + "Linea " + linea + " de Header"

      * Borra la línea de header
      * s = "]" + Se + linea + Se + CHR(127)

      Enviar (s)

next

for i=11 to 20

      linea = alltrim(str(i))

      s = "]" + Se + linea + Se + "Linea " + linea + " de Trailer"

      * Borra la línea de header
      * s = "]" + Se + linea + Se + CHR(127)

      Enviar (s)

next

RETURN 0
```

```
* ****
* ** Funcion: ImprimirTicket
* ****

FUNCTION ImprimirTicket

      s = "@" + Se + "T" + Se + "T"
      Enviar (s)

      s = "A" + Se + "Texto Fiscal" + Se + "0"
      Enviar (s)

      s = "B" + Se + "Plu Nro 1" + Se + "5.000" + Se + "100.00" ;
            + Se + "18.00" + Se + "M" + Se + "0.00" + Se + "0" ;
            + Se + "T"
      Enviar (s)

      s = "D" + Se + "Pago Nro 1" + Se + "2222.00" + Se + "T" + Se + "0"
      Enviar (s)

      s = "E"
      Enviar (s)

RETURN 0

* ****
* ** Funcion: ImprimirFactura
* ****

FUNCTION ImprimirFactura

*
* Datos del comprador (Responsable Inscripto)
*

s = "b" + Se + "Juan Perez" + Se + "20183697308" + Se + "I" + Se + "C"
Enviar (s)

*
* Abre un comprobante fiscal de tipo factura A
*

s = "@" + Se + "A" + Se + "T"
Enviar (s)
```

```
*
* Venta de los articulos
*

* Descripcion adicional como texto fiscal
s = "A" + Se + "Descripcion adicional ... " + Se + "0"
Enviar (s)

s = "B" + Se + "Articulo Nro 1" + Se + "5.000" + Se + "100.00" ;
      + Se + "18.00" +  Se + "M" + Se + "0.00" + Se + "0" + Se + "B"
Enviar (s)

s = "B" + Se + "Articulo Nro 2" + Se + "4.000" + Se + "50.00" ;
      + Se + "21.00" +  Se + "M" + Se + "0.00" + Se + "0" + Se + "B"
Enviar (s)

*
* Percepciones a aplicar
*

s = "`" + Se + "*.*" + Se + "Percepcion A" + Se + "20.00"
Enviar (s)

s = "`" + Se + "*.*" + Se + "Percepcion B" + Se + "15.00"
Enviar (s)

*
* Realiza el pago
*

s = "D" + Se + "Pago Nro 1" + Se + "2222.00" + Se + "T" + Se + "0"
Enviar (s)

*
* Cierra el ticket
*

s = "E"
Enviar (s)


RETURN 0

*****
* Funcion: Enviar
*
* Envia un comando al impresor fiscal y analiza la respuesta.
*****

FUNCTION Enviar

PARAMETERS String

PRIVATE Result

? "Comm: " + String
```

```
* Si la funcion MandaPaq retorna un numero menor que cero, retorna.
* Esto puede ser por un problema de comunicaciones con el impresor.

IF (MandaPaq (Handler, String) < 0 )
      ? "Error enviando el comando"
      RETURN -1
ENDIF

* Levanta la respuesta.
Result = Respuesta (Handler)

? "Resp: " + Result

* Analiza si existe algun error.
GetErrors (Result)

? " "

RETURN 0

****
* FUNCTION GetErrors
*
* Esta funcion levanta la respuesta del printer e imprime en
* el mensaje de error si es que existe.
****

FUNCTION GetErrors

PARAMETERS Resp

PRIVATE Origen, OffsetSep, i, c

DECLARE FiscalErrors [16]
DECLARE PrinterErrors[16]

FiscalErrors[1] =       "Error en chequeo de memoria fiscal"
FiscalErrors[2] =       "Error en chequeo de la memoria de trabajo"
FiscalErrors[3] =       "Carga de bateria baja"
FiscalErrors[4] =       "Comando desconocido"
FiscalErrors[5] =       "Datos no validos en un campo"
FiscalErrors[6] =       "Comando no valido para el estado fiscal actual"
FiscalErrors[7] =       "Desborde del total"
FiscalErrors[8] =       "Memoria fiscal llena"
FiscalErrors[9] =       "Memoria fiscal a punto de llenarse"
FiscalErrors[10] =      ""
FiscalErrors[11] =      ""
FiscalErrors[12] =      "Error en ingreso de fecha"
FiscalErrors[13] =      "Recibo fiscal abierto"
FiscalErrors[14] =      "Recibo abierto"
FiscalErrors[15] =      "Factura abierta"
FiscalErrors[16] =      ""
```

```
PrinterErrors[1]  = ""
PrinterErrors[2]  = ""
PrinterErrors[3]  = "Error de Impresora"
PrinterErrors[4]  = "Impresora Offline"
PrinterErrors[5]  = "Falta papel del diario"
PrinterErrors[6]  = "Falta papel de tickets"
PrinterErrors[7]  = "Buffer de Impresora lleno"
PrinterErrors[8]  = ""
PrinterErrors[9]  = ""
PrinterErrors[10] = ""
PrinterErrors[11] = ""
PrinterErrors[12] = ""
PrinterErrors[13] = ""
PrinterErrors[14] = ""
PrinterErrors[15] = ""
PrinterErrors[16] = ""

Origen = 0

OffsetSep = AT ( CHR(28), Resp )

* Convierte en hexa el status del impresor
PrinterStatus = HexaToInt (SUBSTR ( Resp, Origen, OffsetSep - 1))

IF PrinterStatus < 0
      RETURN -1
ENDIF

Origen = OffsetSep

* Analiza los bits comenzando del menos significativo
FOR i = 1 TO 16
      IF ( INT (PrinterStatus % 2) == 1 )
            IF ( LEN (PrinterErrors[i]) > 0 )
                  ? "PrinterStatus: " + PrinterErrors[i]
            ENDIF
      ENDIF
      PrinterStatus = PrinterStatus / 2
NEXT

OffsetSep = AT ( CHR(28), SUBSTR (Resp, Origen + 1) )

IF OffsetSep == 0
      OffsetSep = LEN(Resp)
ENDIF

* Convierte en hexa el status fiscal
FiscalStatus = HexaToInt (SUBSTR (Resp, Origen + 1, OffsetSep - 1))

IF FiscalStatus < 0
      RETURN -1
ENDIF
```

```
* Analiza los bits comenzando del menos significativo
FOR i = 1 TO 16
      IF ( INT (FiscalStatus % 2) == 1 )
            IF ( LEN (FiscalErrors[i]) > 0 )
                  ? "FiscalStatus: " + FiscalErrors[i]
            ENDIF
      ENDIF
      FiscalStatus = FiscalStatus / 2
NEXT

RETURN 0

****
* FUNCTION HexaToInt
*
* Esta funcion convierte un numero hexadecimal en su equivalente
* en binario.
****

FUNCTION HexaToInt

PARAMETERS HexValue

PRIVATE i, Value, Status

Status = 0

FOR i = 4 TO 1 STEP -1

      S = SUBSTR(HexValue, i, 1)

      Value = ASC (S)

      IF ( Value >= ASC("A") .AND. Value <= ASC("F") )

            Value = Value - ASC("A") + 10

      ELSEIF ( Value >= ASC("a") .AND. Value <= ASC("f") )

            Value = Value - ASC("a") + 10

      ELSEIF ( Value >= ASC("0") .AND. Value <= ASC("9") )

            Value = Value - ASC("0")

      ELSE
            ? "HexaToInt: Numero hexadecimal incorrecto: " + HexValue
            RETURN -1
      ENDIF

      Status = Status + Value * (16 ** ( 4 - i ))

NEXT

RETURN Status
```

WINDOWS

Existen dos formas de implementar esto en un programa Windows: mediante un objeto ActiveX o mediante una librería de enlace dinámico (DLL). El objeto ActiveX trabaja solamente en 32 bits, mientras que la DLL posee una versión para cada plataforma. En general, es más recomendable trabajar con el objeto ActiveX si el lenguaje lo soporta (Visual Basic, Delphi, C++ Builder, Visual FoxPro, etc., ver manual aparte). Caso contrario, o que se quiera disponer de un solo fuente para las dos plataformas, se puede usar la DLL, a la que se la llama con los siguientes puntos de entrada:

```
int OpenComFiscal (int Com, int Mode)
```

Abre el puerto especificado en la variable Com. El modo de apertura indica si el comercio de strings es según el estándar ANSI o ASCII. Para el primero, el valor es 1, para el segundo, 0. Devuelve un handler si se pudo abrir el puerto, < 0 si hubo problemas. Este handler hay que usarlo en cada llamada a funciones fiscales.

```
int MandaPaqueteFiscal (int Handler, char *Buffer)
```

Manda un paquete al printer. Se espera un string que tenga el siguiente formato:

```
n[parámetros...]
```

donde

`n`	: número de función (un byte ASCII)
`parámetros`	: lista de parámetros separados por el separador de campos (ASCII 28) (Los parámetros son opcionales al comando que se elija).

Retorna: 0 si todo bien, < 0 si hubo algún error en la transmisión.

```
int UltimoStatus (int Handler, short *FiscalStatus, short *PrinterStatus)
```

Copia el estado del controlador en las variables FiscalStatus y PrinterStatus. Devuelve 0 si todo bien, o < 0 si ocurrió algún error.

```
int UltimaRespuesta (int Handler, char *Buffer)
```

Obtiene la última respuesta que manejó el impresor para ese handler. El buffer debe tener espacio suficiente para la respuesta más grande (512 bytes es suficiente). Tiene el siguiente formato:

```
PrinterStatus<FS>FiscalStatus<FS>Campos...
```

donde:

FS	:	Field Separator, carácter ASCII 28.
PrinterStatus	:	Status del impresor, 4 bytes en formato hexadecimal
FiscalStatus	:	Status fiscal, 4 bytes en formato hexadecimal
Campos	:	Campos de la respuesta fiscal, separados por el carácter ASCII 28. Su longitud depende de cada comando.

Devuelve 0 si todo bien, o menos que 0 si hubo algún error.

```
void CloseComFiscal (int Handler)
```

Cierra el puerto abierto especificado con el handler.

```
int ReOpenComFiscal (int Com)
```

Verifica si el puerto especificado en la variable Com ya se encontraba abierto.
Devuelve el handler del puerto si el mismo ya estaba abierto, o un código de error (< 0) si hubo problemas. Este handler hay que usarlo en cada llamada a funciones fiscales.

```
int InitFiscal (int Handler)
```

Sincroniza los paquetes para evitar repeticiones de paquetes.
Recomendable llamarlo después de abrir el puerto para poner el impresor a un estado conocido.

```
int VersionDLLFiscal (void)
```

Devuelve el número de la versión de la DLL.

```
void BusyWaitingMode(int Mode)
```

Setea el modo de trabajo de la DLL. Si `Mode` es 0, la DLL cederá la CPU en los momentos de espera. De otra forma, la DLL no retornará hasta que el proceso se haya terminado. Este último modo es el default, y el más seguro, pero también es el más lento. Si se usa el modo 0, hay que asegurar por medio del programa que lo usa que dos paquetes no se superpongan para un mismo handler.

```
int CambiarVelocidad(int PortNumber, long NewSpeed)
```

Cambia la velocidad de transmisión, para aquellos controladores que soporten más velocidad. El máximo aceptado es 57600 baudios.

```
long SearchPrn(int Handler)
```

Realiza la busqueda del Controlador Fiscal en el Puerto asociado a Handler, en las siguientes velocidades: 1200, 2400, 4800, 9600, 19200, 38400 y 57600 baudios.
Devuelve la velocidad a la que se sincronizaron, o un código de error (< 0) si el Controlador Fiscal no fue encontrado.

```
void ProtocolMode(int Mode)
```

Elige entre los distintos protocolos, a saber: OLD_PROTOCOL (0) o NEW_PROTOCOL (1). En el nuevo, se trabaja con la modalidad de comandos STAT_PRN (ver apéndice).

```
int SetCommandRetries (int Retries)
```

Configura el número de reintentos máximo de transmisiones de paquetes fiscales transmitidos por la función 'MandaPaqueteFiscal' cuando el Controlador Fiscal no responde (Time Out).
Devuelve el número de reintentos previo a la re-configuración.

```
void Abort (int PortNumber)
```

Interrumpe (aborta) el proceso en curso correspondiente al Puerto especificado en PortNumber.
Se utiliza para abortar procesos que se encuentran en estado de espera hasta que se cumpla un cierto Time Out.

Tabla de Errores devueltos por las funciones:

-1: Error general
-2: Handler inválido
-3: Intento de enviar un comando cuando se estaba procesando.
-4: Error de comunicaciones.
-5: Puerto ya abierto.
-6: No hay memoria
-7: El puerto ya estaba abierto
-8: La dirección del buffer de respuesta es inválida.
-9: El comando no finalizó, sino que volvió una respuesta tipo STAT_PRN.
-10: El proceso en curso fue abortado por el usuario.

Detección de Problemas:

Dentro de las DLL, existe un mecanismo que permite el rastreo problemas, que consiste en guardar en un archivo todas las operaciones realizadas con las DLL, junto con los detalles de la comunicación. Esto permite seguir el protocolo "de cerca". Para usar esta opción hay que especificar un par de variables de ambiente antes de correr la DLL. Estas variables son:

- `FILELOG`: Especifica el archivo donde se va a guardar la información. Si este archivo no se especifica, no se guardará la información.
- `DEBUGLEVEL`: Nivel de detalle de la información. Es un número de 1 a 3, donde :
  - 1: Sólo la información estrictamente necesaria.
  - 2: Información complementaria a la anterior
  - 3: Toda la información

  Si esta variable no se especifica, se asume 3.

ANSI o ASCII?

El impresor fiscal trabaja con el juego de caracteres ASCII según el cual a la letra "ñ" le corresponde la cifra 164. Dentro de Windows, el juego de caracteres es el ANSI, según el cual es el número 241 el que equivale a la letra "ñ". Ambos juegos difieren a partir del carácter 7F hexadecimal (127 decimal). Para evitar divergencias, se provee una conversión interna que salva este problema. Si la apertura se realiza con la constante MODE_ANSI (1), se convierten todos los strings, tanto los que van al printer como los que vuelven. En el otro modo (MODE_ASCII, 0), los strings viajan tal cual son. La única excepción en la conversión es el carácter F4 hexadecimal, por corresponder a la especificación de doble ancho. Este carácter no será convertido a pesar del modo en que esté abierta la comunicación. F4 hexadecimal corresponde al carácter ANSI **ô**.

Modo de Operación:

La forma correcta de operación debería seguir la siguiente secuencia:

Inicialización	`OpenComFiscal()` Loop hasta que `InitFiscal()` devuelva OK.
Programa Principal	Si se está trabajando sin BusyWaiting, (modo 0), controlar posibilidad de interferencias. `MandaPaqueteFiscal()` con control de errores. Recuperación de estados del printer con `UltimoStatus()`. Chequeo de errores del impresor con los estados recuperados Posible uso de la respuesta con `UltimaRespuesta()`.
Clean Up	`CloseComFiscal()`.

Ejemplo de aplicación en Windows:

El siguiente es un ejemplo de aplicación en Visual Basic (16 Bits). El programa abre el COM 1 como fiscal y cambia la fecha del controlador fiscal. Los Chr$(28) son los separadores de campo.

NOTA: La declaración de los prototipos de las funciones exportables varía de acuerdo a cada lenguaje. Tener en cuenta que, por ejemplo, en Visual Basic la traducción del tipo "int" es diferente para cada plataforma (WIN16 o WIN32).
La DLL deberá ser copiada a \Windows\System, algún directorio accesible por PATH o el mismo directorio de la aplicación. Existe una versión para cada una de las dos plataformas (16 y 32 bits), y sus nombres son WinFis16.DLL y WinFis32.DLL respectivamente.

```
'PROGRAMA DEMOSTRACION DLL FISCAL ESCRITO EN VB 4.0 16 Bits

Declare Function MandaPaqueteFiscal Lib "WinFis16" _
      (ByVal Handler As Integer, ByVal Name As String) As Integer
Declare Function OpenComFiscal Lib "WinFis16" _
      (ByVal Puerto As Integer, ByVal Mode As Integer) As Integer
Declare Function InitFiscal Lib "WinFis16" _
      (ByVal Handler As Integer) As Integer
Declare Function UltimoStatus Lib "WinFis16" _
      (ByVal Handler as Integer, ByRef FiscalStatus as Integer, _
      ByRef PrinterStatus as Integer) as Integer

Const MODE_ANSI = 1
Dim Handler As Integer

Sub EnviarStringFiscal(Comando As String)

    Dim PrinterStatus As Integer, FiscalStatus As Integer

    If MandaPaqueteFiscal(Handler, Comando) < 0 Then
      Screen.MousePointer = 0
      MsgBox "Error mandando paquete"
    else
      If UltimoStatus (Handler, FiscalStatus, PrinterStatus) = 0 Then
            MsgBox "PrinterStatus: " & PrinterStatus & _
                   "FiscalStatus: " & FiscalStatus, vbOkOnly, "Status"
      End If
    End If

End Sub

Sub OpenPort (Port As Integer)

    Handler = OpenComFiscal(Port, MODE_ANSI)
    If Handler < 0 Then
      MsgBox "No pude abrir puerto " & Port
      End
    End If

End Sub

Sub Form_Load ()

    OpenPort 1
    If InitFiscal(Handler) < 0 Then
        MsgBox "Falló InitFiscal()"
        End
    End If

    EnviarStringFiscal "X" & Chr$(28) & "971231" & Chr$(28) & "235900"

End Sub
```

Manejo de la DLL en modo BUSYWAITING_OFF:

La DLL manejada en este modo tiene la ventaja de poder detectar problemas de comunicación o de falta de papel desde la aplicación que la usa. Si el impresor se queda sin papel en medio de un comando, no responderá hasta que se normalice la situación. En estos casos, la DLL no devuelve el control, y el aplicativo parece que está "colgado". Para avisar al operador del problema, podría dispararse un timer que a *n* segundos le avise que algo está ocurriendo. Por ejemplo, en Visual Basic:

```
Sub Mandar(Comando As String)

    Dim rc As Integer

    Screen.MousePointer = vbHourglass
    Timer.Enabled = True
    Do
        rc = MandaPaqueteFiscal(Handler, Comando)
        If rc < 0 And rc <> ERR_ATOMIC Then
            MsgBox "Error " & rc & " mandando paquete"
            Screen.MousePointer = vbNormal
            Timer.Enabled = False
            Exit Sub
        End If
        DoEvents
    Loop Until rc = 0

    Timer.Enabled = False
    Screen.MousePointer = vbNormal

End Sub

Private Sub Timer_Timer()
    'Mostrar un mensaje de advertencia...
    ShowWarning.Show
End Sub
```

El formulario "ShowWarning" es del tipo DialogModal y contiene un mensaje como "Controle el impresor". Tiene un timer que al segundo y medio descarga el formulario para permitir la ejecución de la DLL otra vez. El timer del formulario principal se puede activar en un lapso de cinco segundos.
La DLL tiene que estar funcionando en el modo BUSYWAITING_OFF, obviamente. En este modo, hay que controlar que no se superpongan comandos, por lo cual hay que proveer cierta capa de protección:

```
Sub Mandar (Comando as String)
      Static Atomic as Boolean

      If Atomic Then
          MsgBox "No puede ejecutar más de un comando por vez"
          Exit Sub
      End If

      Atomic = True

      'Proceso real de Mandar...
      ...

      Atomic = False
End Sub
```

Acerca de las declaraciones en distintos lenguajes :

Junto con la DLL en los discos de distribución, se adjuntan módulos de declaración de la funciones escritas para Visual Basic en sus dos modelos de trabajo (16 y 32 bits). De todas formas, es útil recordar que lo que en 16 bits se escribe como `Integer`, en 32 bits es `Long`. Esto es porque en la declaración de una DLL, el entero adopta el tamaño de la palabra del sistema operativo, es decir, 16 o 32 bits. En Visual Basic, el tamaño de un entero es indiferente a su plataforma; un `Integer` siempre tendrá 16 bits, y un `Long` 32 bits. Esto hace que las definiciones de funciones exportables idénticas en operación varíen al cambiar la plataforma.
En cuanto a los demás lenguajes, se ha de buscar en la documentación los equivalentes a los tipos de C, particularmente cómo pasar argumentos por valor o por referencia, y discernir los tamaños de los distintos tipos de entero. Normalmente esta información se asocia a las llamadas a la API de Windows, donde se ha de acceder a DLLs del sistema.

ActiveX:

Una de las principales ventajas asociadas a los objetos ActiveX es justamente liberar al usuario de la especificación de prototipos, típicas fuentes de error. El lenguaje de programación (o su entorno) "entiende" la DLL ActiveX y puede formular el código necesario para acceder a ella de forma segura. Esto permite que el objeto ActiveX sea mucho más completo que una DLL: puede proporcionar cada comando del controlador sin preocuparse por el tipo de sus parámentros, resuelto directamente por el lenguaje. Por tanto, el objeto ActiveX para manejar el controlador fiscal contiene mucho más inteligencia y funcionalidad que la DLL. Ver manual aparte.

Ejemplos:

A continuación se detalla un ejemplo según FoxPro 16 bits, FoxPro de 32 bits, luego en Clarion 16 bits, y finalmente en Delphi 3.0 (32 bits):

```
*
* Ejemplo de implementación DLL fiscal
* en FoxPro 2.5 16 bits usando FoxTools.

* Declaraciones de la DLL:
*
set library to "foxtools.fll" addi
pOpenComFiscal = regfn("OpenComFiscal", "II", "I", "WINFIS16.DLL")
pCloseComFiscal = regfn("CloseComFiscal", "I", "", "WINFIS16.DLL")
pMandaPaqueteFiscal=regfn("MandaPaqueteFiscal", "I@C","I","WINFIS16.DLL")
pUltimaRespuesta = regfn("UltimaRespuesta", "I@C", "I", "WINFIS16.DLL")
pUltimoStatus = regfn("UltimoStatus", "I@I@I", "I", "WINFIS16.DLL")
pVersionDLLFiscal = regfn("VersionDLLFiscal", "", "I", "WINFIS16.DLL")
pInitFiscal = regfn("InitFiscal", "I", "I", "WINFIS16.DLL")
pBusyWaitingMode = regfn("BusyWaitingMode", "I", "", "WINFIS16.DLL")

Handler=callfn (pOpenComFiscal, 1, 1) && COM1, ANSI
if Handler < 0
      * Problema de puertos: no instalado, en uso, etc.
endif
if callfn (pInitFiscal,Handler) < 0
      * Problema de comunicaciones: impresor apagado,
      * no responde, cables, etc.
endif

* Apertura de Ticket.
Comando = "@" + chr(28) + "T" + chr(28) + "T"
if callfn (pMandaPaqueteFiscal, Handler, @Comando) < 0
      * Para cada llamada a pMandaPaqueteFiscal
      * si devuelve < 0 es problemas de comunicaciones.
      * Además, hay que comprobar la respuesta del controlador.
endif

* Respuesta del controlador:
Respuesta=space(512)
=callfn (pUltimaRespuesta, Handler, @Respuesta)

* En Respuesta se tiene la respuesta del controlador.
* Ver ejemplo para FoxPro DOS acerca de cómo tratarla.
* Esto hay que hacerlo para cada llamada a pMandaPaqueteFiscal.

* Línea de venta
Comando= "B" + chr(28) + "Artículo" + chr(28) + "1.00";
            + chr(28) + "10.0" + chr(28) + "21.00" ;
            + chr(28) + "M" + chr(28) + "0.0" ;
            + chr(28) + "1" + chr(28) + "T"

= callfn (pMandaPaqueteFiscal, Handler , @Comando)

* Medio de Pago
Comando = "D" + chr(28) + "Efectivo" + chr(28) + "10.00";
            + chr(28) + "T" + chr(28) + "1"
=callfn (pMandaPaqueteFiscal, Handler, @Comando)

* Cierre de Ticket.
Comando = "E"
= callfn (pMandaPaqueteFiscal, Handler, @Comando)

* Cierre del puerto.
callfn (pCloseComFiscal, Handler);
```

```
* Ejemplo en FoxPro 5.0 32 bits
* Este ejemplo realiza un ticket analizando los errores a medida
* que se van enviando los comandos. Para realizar una FACTURA se
* recomienda mirar el ejemplo de FOX para DOS.

DECLARE INTEGER UltimaRespuesta IN WinFis32.dll ;
            AS UltimaRespuesta ;
            INTEGER nHandler , STRING  @ cBuffer
DECLARE INTEGER OpenComFiscal IN WinFis32.dll ;
            AS OpenComFiscal ;
            INTEGER nCom , INTEGER nModo
DECLARE INTEGER InitFiscal IN WinFis32.dll ;
            AS InitFiscal ;
            INTEGER nHandler
DECLARE CloseComFiscal IN WinFis32.dll ;
            AS CloseComFiscal ;
            INTEGER nHandler
DECLARE INTEGER MandaPaqueteFiscal IN WinFis32.dll ;
            AS MandaPaqueteFiscal ;
            INTEGER nHandler , STRING  @ cBuffer
DECLARE INTEGER UltimoStatus IN WinFis32.dll ;
            AS UltimoStatus ;
            INTEGER nHandler ;
            , INTEGER @ nFiscalStatus , INTEGER @ nPrinterStatus
DECLARE INTEGER VersionDLLFiscal IN WinFis32.dll ;
            AS VersionDLLFiscal
DECLARE BusyWaitingMode IN WinFis32.dll ;
            AS BusyWaitingMode ;
            INTEGER nMode

M.oFormulario = CREATE( 'Form' )

M.oFormulario.AddObject( 'ImpresorFiscal' , 'ClassImpresorFiscal' )

      WITH M.oFormulario.ImpresorFiscal

            * Abre e inicializa el port de comunicaciones
            .OpenComFiscal( )

            * Cancela un ticket que haya quedado por la mitad
            .CancelTicket ()

            * Cierra el ticket que no pudo ser cerrado
            .CloseTicket ()

            * Imprime un ticket
            .PrintTicket ()

            * Cierra el port de comunicaciones
          .CloseComFiscal( )

      ENDWITH

RETURN
```

```
DEFINE CLASS ClassImpresorFiscal AS Custom
      nHandler = - 1
      nCom = 1
      nModo = 1
      Se = CHR( 28 )
      cBuffer = ''

      FUNCTION OpenComFiscal

            PRIVATE oEste , nResp
            LOCAL oEste , nResp

            M.oEste = THIS
            M.nResp = 0

            WITH M.oEste

                  M.nResp = OpenComFiscal( .nCom , .nModo )

                  IF M.nResp < 0
                        = .MensajeErrorDLLS( M.nResp, CHR( 13 ) + ;
                              'Al Abrir Com ' + ALLTRIM(STR(.nCom ) ) )
                        RETURN -1
                  ENDIF

                  * Guarda el Handler
                  .nHandler = M.nResp

                  M.nResp = InitFiscal( .nHandler )

                  IF M.nResp < 0
                        = .MensajeErrorDLLS( M.nResp, CHR( 13 )  + ;
                              'Al Iniciar Com ' + ALLTRIM( STR( .nCom )))
                        RETURN -1
                  ENDIF

                  RETURN 0

            ENDWITH

      ENDFUNC
```

```
FUNCTION CloseComFiscal

      PRIVATE oEste , nResp
      LOCAL oEste , nResp

      M.oEste = THIS
      M.nResp = 0

      WITH M.oEste

            IF .nHandler >= 0
                  CloseComFiscal( .nHandler )
                  RETURN 0
            ENDIF

            .nHandler = - 1
            RETURN -1

      ENDWITH

ENDFUNC

FUNCTION MandaPaqueteFiscal

      LPARAMETERS cCadena1

      PRIVATE oEste , cCadena , nResp
      LOCAL oEste , cCadena , nResp

      M.oEste = THIS

      IF UPPER( TYPE( 'M.cCadena1' ) ) == 'C'
            M.cCadena = M.cCadena1
      ELSE
            RETURN (-1)
      ENDIF

      IF M.oEste.nHandler < 0
            RETURN (-1)
      ENDIF

      M.nResp = MandaPaqueteFiscal( M.oEste.nHandler , M.cCadena )

      IF M.nResp < 0
            = M.oEste.MensajeErrorDLLS( M.nResp , CHR( 13 ) + ;
                  'Al Operar Com ' + ALLTRIM( STR( M.oEste.nCom )))
      ENDIF

      RETURN M.nResp

ENDFUNC
```

```
* Cancela un Ticket
* La respuesta al comando puede dar Comando Inválido si el impresor
* realmente NO tiene un ticket Abierto

FUNCTION CancelTicket
      s = "D" + .Se + " " + .Se + "0.00" + .Se + "C" + .Se + "0"
      RETURN .Enviar (s)
ENDFUNC

* Cierra un Ticket
* La respuesta al comando puede dar Comando Inválido si el impresor
* realmente NO tiene un ticket Abierto

FUNCTION CloseTicket
      s = "E"
      RETURN .Enviar (s)
ENDFUNC

* Imprime un ticket Completo chequeando errores

FUNCTION PrintTicket

      PRIVATE oEste , nResp
      LOCAL oEste , nResp, s

      M.oEste = THIS
      M.nResp = 0

      WITH M.oEste

            * Abre el ticket
            s = "@" + .Se + "T" + .Se + "T"
            if .Enviar (s) < 0
                  ? "Error enviando el comando: " + s
            endif

            * Imprime un texto no fiscal
            s = "A" + .Se + "Texto Fiscal" + .Se + "0"
            if .Enviar (s) < 0
                  ? "Error enviando el comando: " + s
            endif

            * Vende un producto
            s = "B" + .Se + "Plu Nro 1" + .Se + "5.000" + .Se ;
                  + "100.00" + .Se + ;
                  "18.00" +   .Se + "M" + .Se + "0.11" + .Se + ;
                  "0" + .Se + "T"
            if .Enviar (s) < 0
            ? "Error enviando el comando: " + s
            endif

            * Realiza el pago
            s = "D" + .Se + "Pago Nro 1" + .Se + "2222.00" + ;
                  .Se + "T" + .Se + "0"
            if .Enviar (s) < 0
                  ? "Error enviando el comando: " + s
            endif

            * Cierra el ticket
```

```
            s = "E"
            if .Enviar (s) < 0
                  ? "Error enviando el comando: " + s
            endif

      ENDWITH

ENDFUNC

FUNCTION Enviar

      LPARAMETERS Comando
      PRIVATE MyBuffer, nResp
      PRIVATE oEste , nResp
      PRIVATE FiscalStatus, PrinterStatus

      FiscalStatus  = 0
      PrinterStatus = 0

      M.MyBuffer = REPLICATE (' ' , 520 )

      M.oEste = THIS
      M.nResp = 0

      WITH M.oEste

            * Envia el comando al impresor.
            M.nResp = .MandaPaqueteFiscal (Comando)

            IF M.nResp < 0
                  RETURN M.nResp
            ENDIF

            * Recupera la respuesta Completa al comando
            * M.nResp = UltimaRespuesta( .nHandler , @MyBuffer )
            * ? MyBuffer

            * Recupera sólo los status fiscal y
            * de printer de la respuesta.
            M.nResp = UltimoStatus( .nHandler , @FiscalStatus, ;
                                    @PrinterStatus )

            IF M.nResp < 0
                  RETURN M.nResp
            ENDIF

            * Analiza la respuesta y retorna
            * lo que devuelve CheckErrors luego de
            * la verificación de errores

            RETURN .CheckErrors (FiscalStatus, PrinterStatus)

      ENDWITH


ENDFUNC
```

```
*********************************************************************
* Esta funcion chequea la respuesta del printer e imprime el
* mensaje de error si es que existe.
* NOTA: GetErrors DEBE SER INVOCADA POR CADA COMANDO QUE SE ENVIE,
* YA QUE ES LA UNICA FORMA DE NO PERDER EL CONTROL DEL PROGRAMA
* EN CASO DE ERROR!!!!
* Ejemplo de errores: Comando Desconocido, Campo invalido en
* comando.... etc.
*********************************************************************

FUNCTION CheckErrors

      LPARAMETERS FiscalStatus, PrinterStatus

      PRIVATE Error, i, c

      DECLARE FiscalErrors [16]
      DECLARE PrinterErrors[16]

      FiscalErrors[1] = "Error en chequeo de memoria fiscal"
      FiscalErrors[2] = "Error en chequeo de la memoria de trabajo"
      FiscalErrors[3] = "Carga de bateria baja"
      FiscalErrors[4] = "Comando desconocido"
      FiscalErrors[5] = "Datos no validos en un campo"
      FiscalErrors[6] = "Comando invalido p/estado fiscal actual"
      FiscalErrors[7] = "Desborde del total"
      FiscalErrors[8] = "Memoria fiscal llena"
      FiscalErrors[9] = "Memoria fiscal a punto de llenarse"
      FiscalErrors[10] = ""
      FiscalErrors[11] = ""
      FiscalErrors[12] = "Error en ingreso de fecha"
      FiscalErrors[13] = "Recibo fiscal abierto"
      FiscalErrors[14] = "Recibo abierto"
      FiscalErrors[15] = "Factura abierta"
      FiscalErrors[16] = ""

      PrinterErrors[1]  = ""
      PrinterErrors[2]  = ""
      PrinterErrors[3]  = "Error de Impresora"
      PrinterErrors[4]  = "Impresora Offline"
      PrinterErrors[5]  = "Falta papel del diario"
      PrinterErrors[6]  = "Falta papel de tickets"
      PrinterErrors[7]  = "Buffer de Impresora lleno"
      PrinterErrors[8]  = ""
      PrinterErrors[9]  = ""
      PrinterErrors[10] = ""
      PrinterErrors[11] = ""
      PrinterErrors[12] = ""
      PrinterErrors[13] = ""
      PrinterErrors[14] = ""
      PrinterErrors[15] = ""
      PrinterErrors[16] = ""

      Error = 0

      * Analiza los bits comenzando del menos significativo
      FOR i = 1 TO 16
            IF ( INT (FiscalStatus % 2) == 1 )
                  IF ( LEN (FiscalErrors[i]) > 0 )
```

```
                        * Analizar los errores
                        ? "FiscalStatus: " + FiscalErrors[i]
                        IF i < 12
                              * Hasta el bit nro 12 incluído son
                              * condiciones de error reales
                              Error = 1
                        ENDIF
                  ENDIF
            ENDIF
            FiscalStatus = FiscalStatus / 2
      NEXT

      * Analiza los bits comenzando del menos significativo
      FOR i = 1 TO 16
            IF ( INT (PrinterStatus % 2) == 1 )
                  IF ( LEN (PrinterErrors[i]) > 0 )

                        * Analizar en este caso que hacer en caso de
                        * error; ejemplo: Falta Papel
                        ? "PrinterStatus: " + PrinterErrors[i]

                        * Indicar que hubo Error!

                        Error = 1
                  ENDIF
            ENDIF
            PrinterStatus = PrinterStatus / 2
      NEXT

      IF Error = 1
            RETURN -1
      ENDIF

      RETURN 0

ENDFUNC
```

```
	FUNCTION MensajeErrorDLLS

		LPARAMETERS nError , cMensaje2

		PRIVATE cMensaje , cMensaje3
		LOCAL cMensaje , cMensaje3

		IF UPPER( TYPE( 'M.nError' ) ) == 'N'

			IF UPPER( TYPE( 'M.cMensaje2' ) ) == 'C'
				M.cMensaje3 = M.cMensaje2
			ELSE
				M.cMensaje3 = ''
			ENDIF

			DO CASE
				CASE M.nError == - 1
					M.cMensaje = 'Error general'
				CASE M.nError == - 2
					M.cMensaje = 'Handler Inválido'
				CASE M.nError == - 3
					M.cMensaje = 'Intento de Superposición'
				CASE M.nError == - 4
					M.cMensaje = 'Error de Comunicación'
				CASE M.nError == - 5
					M.cMensaje = 'Puerto ya Abierto'
				CASE M.nError == - 6
					M.cMensaje = 'No hay Memoria'
				OTHERWISE
					M.cMensaje = 'Error Desconocido'
			ENDCASE

			M.cMensaje = M.cMensaje + '(' + ;
					ALLTRIM(STR(ABS(M.nError))) + ')' + ;
					'  ' + M.cMensaje3

			= MESSAGEBOX (M.cMensaje , 0 + 48 + 0 , ;
					'Mensaje de Error en DLL Fiscal')

		ENDIF

	ENDFUNC

ENDDEFINE
```

En Clarion 16 bits (versión 200x) se ha de definir un módulo de DLL externo con el siguiente módulo asociado (winfis.clw):

```
      OpenComFiscal(SHORT, SHORT),PASCAL,SHORT
      CloseComFiscal(SHORT),PASCAL
      MandaPaqueteFiscal(SHORT, *CSTRING),PASCAL,RAW,SHORT
      UltimaRespuesta(SHORT, *CSTRING),PASCAL,RAW,SHORT
      VersionDLLFiscal(),PASCAL,SHORT
      InitFiscal(SHORT),PASCAL,SHORT
      BusyWaitingMode(SHORT),PASCAL
```

En el lugar donde espera una librería especificar "winfis16.lib", adjunta con la DLL de 16 bits. El siguiente es un ejemplo de su uso:

Primero definir las siguientes variables globales:

```
Handler     SHORT
RetCode     SHORT
Comando     CSTRING(100)
Respuesta   CSTRING(512)
```

El siguiente código le manda un comando de X fiscal al impresor, dejando en la variable "Respuesta" la respuesta del controlador fiscal:

```
Handler=OpenComFiscal(2, 0)
Comando='9<28>X'
RetCode=MandaPaqueteFiscal(Handler, Comando)
RetCode=UltimaRespuesta(Handler, Respuesta)

DO PROCEDURERETURN
```

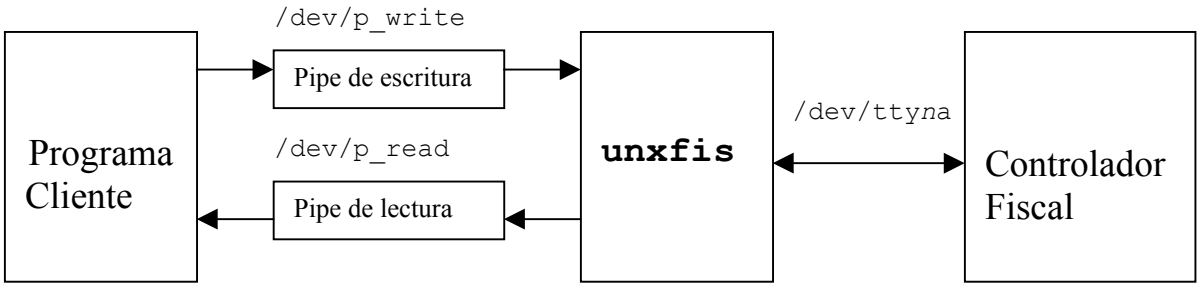
Para Delphi, aquí está la declaración de las funciones exportables de la DLL:

```
type
      pShort = ^smallint;

Function OpenComFiscal (Com, Mode: Integer) : Integer; Far StdCall;
      External 'WinFis32.dll' Name 'OpenComFiscal';
Procedure CloseComFiscal (Handler: Integer); Far StdCall;
      External 'WinFis32.dll' Name 'CloseComFiscal';
Function MandaPaqueteFiscal (Handler: Integer; Buffer: pChar) :
      Integer; Far StdCall; External 'WinFis32.dll'
      Name 'MandaPaqueteFiscal';
Function UltimaRespuesta (Handler: Integer; Buffer: pChar) :
      Integer; Far StdCall; External 'WinFis32.dll'
      Name 'UltimaRespuesta';
Function UltimoStatus (Handler: Integer; FiscalStatus: pShort;
      PrinterStatus:   pShort)   :   Integer;   Far   StdCall;   External
      'WinFis32.dll' Name 'UltimoStatus';
Function VersionDLLFiscal: Integer; Far StdCall; External 'WinFis32.dll'
      Name 'VersionDLLFiscal';
Function InitFiscal (Handler: Integer): Integer; Far StdCall;
      External 'WinFis32.dll' Name 'InitFiscal';
Procedure BusyWaitingMode(Mode: Integer); Far StdCall; External
      'WinFis32.dll' Name 'BusyWaitingMode';
```

LINUX / UNIX SCO Open Server - System V

El programa que servirá para comunicarse con el controlador fiscal es un demonio llamado unxfis. Este demonio deberá correrse en background, preferentemente como uno de los demonios que dispara el sistema al arrancar (instalación por default) o desde el archivo `.profile` de una cuenta. La forma de comunicación es mediante dos pipes según el siguiente esquema:



Para efectuar un comando sobre el impresor fiscal se debe abrir el pipe de escritura (`/dev/p_write` por default) y escribir sobre él como si fuera un archivo. Por cada comando que se escribe se puede recuperar la respuesta del impresor leyendo sobre el pipe de lectura (`/dev/p_read`).

Existe también para Unix una librería para linkear directamente con 'C'. Esta librería evita el uso del unxfis ya que tiene todo el control del protocolo embebido. Más adelante en la sección deterninada para el uso de las librerías se describe su uso.

Instalación:

Para instalar el programa ejecutar los siguientes comandos:

1 - Ingresar en la cuenta de `root`

2 – Cambiar al directorio en donde se va a instalar el programa
Ej:

```
cd /usr/edu
mkdir fiscal
cd fiscal
```

3 - Suponiendo que el diskette de instalación se encuentra en la diskettera A:,
ejecutar los siguientes comandos:

Tanto para Unix como para Linux :

```
doscp -r a:/drivers/unix/unix.tar
tar xvf unix.tar
```

4 – Cambiar al directorio daemon y ejecutar el script `install`:

```
cd daemon
./install
```

Esta operación instala el driver en un directorio a elección para que se dispare entre los demonios del sistema.

Esto en UNIX se logra de tres formas:

a) Generando un archivo en `/etc/conf/init.d` llamado `fis` (es necesario relinkear el kernel para que estos cambios tengan efecto). Para desinstalarlo, ejecutar el programa `uinstall`.
b) Modificar el archivo /etc/rc.d/8/userdef. Este es un archivo que el Unix ejecuta línea por línea, en el momento del arranque. No hace falta relinkear el kernel, y los cambios tienen efecto una vez que se hace shutdown y se arranca nuevamente el sistema.
c) Disparando el programa desde el archivo .profile de la cuenta.

En Linux:

a) Verificar en el manual de la versión de Linux que se dispone la posibilidad de ejecutar programas desde alguno de los directorios /etc/rc.d
b) Disparando el programa desde el archivo .profile de la cuenta.

El programa 'pruf':

Hay un programa denominado `pruf` que al igual que en DOS permite enviar comandos al controlador fiscal y analizar las respuestas a los distintos comandos.
Se ejecuta con la opción –p ttyname, en donde ttyname es el nombre de la tty (sin el path – '/dev/'). Para ver el resto de las opciones de línea de comando, ejecutar el programa sin parámetros y se imprimirá el uso.

Uso del driver:

Para usar el driver, hay que escribir sobre el pipe de escritura un comando fiscal en el siguiente formato:

```
Comando [Sep][Parámetro] [Sep][Parámetro] ......
```

donde:

`Comando:`	Caracter ASCII que corresponde al comando fiscal deseado, por ejemplo, "Y" para pedir la fecha/hora fiscal.
`Sep:`	Caracter separador de campos (hexadecimal 1C o ASCII 28).
`Parámetro:`	Parámetro n del comando. Cada comando tiene una determinada cantidad de parámetros.

Luego se leerá la respuesta desde el pipe de lectura, que podrá ser "-1" en caso de que haya habido un error, o una respuesta fiscal con el siguiente formato:

```
PrinterStatus + Sep + FiscalStatus + Sep + Campos...\n
```

donde:

`PrinterStatus`	:	Status del impresor, 4 bytes en formato hexadecimal
`FiscalStatus`	:	Status fiscal, 4 bytes en formato hexadecimal
`Sep`	:	Separador de campos (ASCII 28).
`Campos`	:	Campos de la respuesta fiscal, separados por el separador de campos. Su longitud depende de cada comando.
`\n`	:	Terminador, carácter ASCII 10 (0Ah).

Implementación:

Las opciones de uso del unxfis se obtienen tipeando el nombre del programa sin opciones en la línea de comando.

```
c:/usr/edu> unxfis <enter>
```

Funcionando con conexión directa al controlador:

Uso: unxfis -p tty [-v] [-t] [-w] [opciones....]

-p tty	tty en la que esta conectado el impresor El programa abre el dispositivo /dev/[tty]
-v bauds	Cambia la velocidad del puerto serie.
-t	Busca al controlador a las velocidades posibles (auto-excluyente con la opcion [-v]).
-w	Establece el uso del nuevo protocolo de comunicaciones con el impresor.

Funcionando como cliente de red:

Uso: unxfis -k -c ipaddr [-r port] [opciones....]

-k	Envia los comandos a un servidor de red.
-r port	Port del servidor de red (default: 1600).
-c ipaddr	Nombre o dirección de ip del servidor.

Opciones generales:

-i ipipe	Nombre del Pipe de escritura. Default: /dev/p_write.
-o opipe	Nombre del Pipe de lectura. Default: /dev/p_read.
-d	Habilita los debuggers en pantalla.
-s	Guarda los debuggers en un archivo (/etc/fiscal.log).
-l file	Cambia el nombre del archivo de debug.
-n	Imprime la versión.
-x	Cambia el separador de campos del string de respuesta (hexadecimal 1C) por el caracter '|'.

Ejemplos de uso :

1) Con conexión directa al puerto serie :

En éste modo, el demonio está directamente conectado con el controlador a través del puerto serie (en el ejemplo */dev/tty1a*). Los pipes por default son */dev/p_write* (de donde lee los comandos) y */dev/p_read* (en donde escribe las respuestas). De ser necesario la utilización de otro juego de pipes, explicitarlos en la línea de comandos con las opciones *–i ipipe y –o opipe.*

```
/usr/edu> unxfis –p tty1a –d
```

El ejemplo ejecuta el demonio con debuggers habilitados en pantalla. Estos debuggers muestran información explicativa del protocolo y es almacenada también en el archivo /etc/fiscal.log. En el caso que se quiera cambiar el nombre de éste archivo se debe invocar al unxfis con la opción *–l* y un nombre de archivo :

```
/usr/edu> unxfis –p tty1a –d –l archivo.log
```

La opción *–x* cambia el separador tradicional por el caracter ´|´ Esto es útil cuando se quiere imprimir la respuesta en pantalla, ya que el Unix no muestra los caracteres de control en la salida estandard.

Con la opción *–w* se habilita el modo de trabajo en el cual el impresor puede reportar un estado de error a través del comando STATPRN :

```
/usr/edu> unxfis –p tty1a –d –l archivo.log -w
```

En éste modo el impresor informa al unxfis que no puede imprimir porque está en estado de error. Luego de enviar el comando a través del pipe de escritura, en lugar de recibir la respuesta al comando original, la aplicación recibe la siguiente respuesta por parte del demonio :

```
PrinterStatus + Sep + FiscalStatus + Sep + "~STATPRN~"
```

Con el objetivo de obtener la respuesta original, la aplicación debería enviar el comando STATPRN (ver manual de comandos) y mientras sigue recibiendo la respuesta mencionada, vuelve a enviar STATPRN hasta que la causa de error desaparece (el impresor se puso Online, se puso el papel, etc). El objetivo principal es que la aplicación pueda enviar un mensaje al usuario indicando el tipo de error en el que está parado el impresor.

Este modo se puede probar con el ´pruf´ si antes de enviar un comando, se pone al impresor en Offline. El pruf imprime un mensaje de error y, si se presiona una tecla, el comando STATPRN es enviado pudiendo observarse que el error desaparece cuando el impresor se pone Online.

2) Funcionando como cliente de red:

En éste modo, el unxfis envia los comando a un servidor de red (típicamente el spooler) cuya dirección de IP (o nombre de host) se pasa como parámetro:

```
/usr/edu> unxfis –k –c "192.0.2.100"
ó
/usr/edu> unxfis –k –c spooler_server
```

El servidor de red que se puede utilizar es el spooler, que utiliza el servicio del port 1600.
En caso que el spooler escuche comandos en otro port, se debe invocar al unxfis con la opción *–r port :*

```
/usr/edu> unxfis –k –c spooler_server –r 1700
```

El funcionamiento del unxfis se puede visualizar fácilmente :

En una tty ejecutamos el unxfis ,

```
unxfis –p tty1a –d
```

y en otra abrimos el pipe de lectura con el comando de Unix ´cat´:

```
cat < /dev/p_read
```

Cambiamos una vez más a una tercera `tty` para ejecutar:

```
cat scriptfile > /dev/p_write
```

donde `scriptfile` es un script ASCII conteniendo comandos válidos para el impresor, por ejemplo una X fiscal:

```
9^\X
```

("^\" es el separador de campos ASCII 28. Para conseguirlo desde el VI, por ejemplo, habría que primero pulsar `CONTROL-V`, y luego `CONTROL-\`).

Cuando se usan los pipes dentro de un lenguaje, una vez usados es conveniente cerrarlos. Si el programa hace un `exit()` o termina, UNIX se encargará de cerrarlos de todos modos.

En caso de que el lenguaje de programación que se use no sea capaz de comerciar con pipes de manera adecuada, se dispone de una herramienta adicional llamada *spooler*, que es capaz de tomar y contestar comandos directamente desde archivos ASCII reales. Consultar la documentación de este programa para más información.

Volviendo al `unxfis`, veamos una ejemplo de aplicación escrita en lenguaje C, ejemplificando su uso dentro de un lenguaje de programación:

EJEMPLO DE USO DE LOS PIPES EN LENGUAJE ´C´:

```
#ifdef UNIX
#include <prototypes.h>
#endif

#include <unistd.h>

#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>

#define FS              0x1c         // ASCII 28
#define CMD_STATPRN 0xa1


void
Error (char *Msg, ... )
{
    va_list argptr;
    char buffer[200];

    va_start (argptr, Msg);
    vsprintf (buffer, Msg, argptr);
    printf ("%s\n", buffer);
    va_end (argptr);

    exit (1);
}

Show (char *Msg, char *buf)
{
       int c;

       printf ("%s", Msg);

       while ( *buf )
       {
              if ( (c = *buf++) == FS )
                     putchar ('|');
              else putchar ( c );
       }
}

int
main (int argc, char *argv[])
{
       FILE *fp;
       char buffer[200], *p;
       int fdpipe_command;
       int fdpipe_status, n;

       if ( argc != 2 )
              Error ("Uso: ftest <file>\n");

       if ( !(fp = fopen (argv[1], "r")) )
              Error ("Error abriendo %s\n", argv[1]);

       if ( (fdpipe_command = open ("/dev/p_write", O_WRONLY)) < 0 )
              Error ("Error abriendo pipe de escritura");

       if ( (fdpipe_status = open ("/dev/p_read", O_RDONLY)) < 0 )
              Error ("Error abriendo pipe de lectura");

       while ( fgets (buffer, sizeof (buffer), fp) )
       {
              Show ("Envio: ", buffer);

              write (fdpipe_command, buffer, strlen (buffer));

              n = read (fdpipe_status, buffer, sizeof (buffer));

              // Si el string STATPRN viene como parte del mensaje de
              // respuesta (al final), es que el impresor no puede imprimir
              // por falta de papel, error mecanico o no esta OnLine.

              // Esta no es la respuesta al comando que se mando. Es una
```

```
                // respuesta de error en la que el usuario puede enviar
                // un mensaje al operador, indicando que solucione el
                // problema del impresor.

                // Esta logica es valida solo para el protocolo nuevo
                // e impresores nuevos, y evita que el programa quede
                // bloqueado, esperando al impresor que salga de su estado
                // de error.

                // Una vez recuperado, el impresor manda la respuesta
                // original

                if ( strstr (buffer, "STATPRN") )
                {
                        do
                        {
                                sleep (2);

                                printf (" *** Error de Impresor *** \n");

                                sprintf (buffer, "%c\n", CMD_STATPRN);

                                // Envia el comando CMD_STATPRN que encuesta al
                                // impresor.
                                // Si este salio de su estado de error, envia la
                                // respuesta al comando original.

                                write (fdpipe_command, buffer, strlen (buffer));

                                // Leo la respuesta.

                                n = read (fdpipe_status, buffer, sizeof(buffer));

                                if (strlen(buffer) == 2 && strstr(buffer, "-1") )
                                {
                                        printf ("Error de comunicacion con el"
                                                        "impresor \n");
                                        exit (1);
                                }
                        }

                        while ( strstr (buffer, "STATPRN") );
                }

                if ( strlen (buffer) == 2 && strstr (buffer, "-1") )
                {
                        printf ("Error de comunicacion con el impresor \n");
                        exit (1);
                }

                Show ("Status: ", buffer);
        }

        fclose (fp);

        close (fdpipe_status);
        close (fdpipe_command);
}
```

Ejemplo en Cobol para Unix

Muestra como abrir un **pipe**, escribir y leer de él. No muestra como imprimir un comprobante, sin embargo soluciona un tema que resulta consulta frecuente.

NOTA:
Para quienes se basen en este ejemplo se acalara que no es necesario abrir y cerrar el pipe por cada string (respuesta / comando) que se desee leer / escribir.
Se abre al arrancar el programa y se cierra al terminar la aplicación.

```
 IDENTIFICATION DIVISION.
*****************************************************************************************************
 * This program is provided for demonstration and educational purposes only by Ryan
 * McFarland Corporation (a division of Liant Software).
* It is neither supported nor warranted by Ryan McFarland Corporation.
* (c) 1992, 1995 Ryan McFarland Corporation.  All rights reserved.
*****************************************************************************************************
 PROGRAM-ID.  rp.
 AUTHOR.  RAH.
 DATE-COMPILED.
 REMARKS.
        This program will read from a pipe file.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  RMCOBOL.
 OBJECT-COMPUTER.  RMCOBOL.
 SPECIAL-NAMES.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
    SELECT IN-FILE ASSIGN TO INPUT, "pipefile"
         ORGANIZATION IS LINE SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  IN-FILE.
 01  IN-REC                    PIC X(40).
 WORKING-STORAGE SECTION.
 01  Count-Line.
    03  Filler    PIC X(15) VALUE "Records Read = ".
    03  Rec-Count PIC 9(5)  VALUE 0.
 PROCEDURE DIVISION.
 A.
     OPEN INPUT IN-FILE.
 LOOP-MODULE.
     READ IN-FILE
        AT END

            GO TO END-IT.
     ADD 1 TO REC-COUNT.
     DISPLAY IN-REC POSITION 1.
     MOVE SPACES TO IN-REC.
```

```
    GO TO LOOP-MODULE.
END-IT.
    CLOSE IN-FILE.
    DISPLAY COUNT-LINE.
    STOP RUN.
END PROGRAM RP.
```

La librería para LINUX / UNIX

La librería para Unix (fislib.a) tiene los mismos puntos de entrada que la librería para DOS:

- `int OpenCommFiscal (char *PortName);`

Abre el puerto físico determinado por 'PortName'. Este es el nombre de la tty a ser usada (tty1a, tty2a, etc....) sin empezar con /dev. Si el puerto no se puede abrir, por alguna razón, devuelve –1. Si todo estuvo bien, devuelve un descriptor que se usará para los demás puntos de entrada.

- `int MandaPaqueteFiscal (int PortDescriptor, char *Command,`
  `unsigned short *FiscalStatus,`
  `unsigned short *PrinterStatus, char *AnswerBuffer);`

Manda un paquete al puerto de comunicaciones. `PortDescriptor` es el descriptor devuelto por `OpenCommFiscal`. El parámetro `Command` es el comando a enviar, que debe tener el siguiente formato:

`Comando [Sep][Parámetro] [Sep][Parámetro] ......`

donde:

`Comando:`	Caracter ASCII que corresponde al comando fiscal deseado, por ejemplo, "Y" para pedir la fecha/hora fiscal.
`Sep:`	Caracter separador de campos (hexadecimal 1C o ASCII 28).
`Parámetro:`	Parámetro n del comando. Cada comando tiene una determinada cantidad de parámetros.

`AnswerBuffer` es un puntero a un buffer con capacidad suficiente para albergar una respuesta fiscal con el siguiente formato:

`PrinterStatus + Sep + FiscalStatus + Sep + Campos...`

donde:

`PrinterStatus:`	Status del impresor, 4 bytes en formato hexadecimal
`FiscalStatus :`	Status fiscal, 4 bytes en formato hexadecimal
`Sep :`	Separador de campos (ASCII 28)
`Campos :`	Campos de la respuesta fiscal, separados por el separador de campos. Su longitud depende de cada comando.

Un buffer de 500 bytes es suficiente para la respuesta más larga. Los punteros `FiscalStatus` y `PrinterStatus` deben apuntar a dos variables de tipo `unsigned short` para guardar los dos estados del controlador fiscal. Esta función devolverá < 0 si algo ha ido mal, y 0 si todo estuvo bien. Consultar la tabla de errores en la sección Windows. En caso que devuelva < 0, los datos devueltos en los punteros pueden contener cualquier cosa.

- `int CloseCommFiscal (int PortDescriptor);`

Cierra las comunicaciones. El parámetro `PortDescriptor` es el devuelto por `OpenCommFiscal`.

- `int InitFiscal (int PortDescriptor);`

Sincroniza al impresor con la numeración de paquetes que lleva la librería

- `int SetKeepAliveHandler (void (*f)());`

Función de Espera: Instala una función a ser ejecutada cuando el impresor está ocupado, o está parado por falta de papel. En los impresores nuevos que soportan STATPRN se sugiere habilitar el nuevo protocolo. Esto hace que la función MandaPaqueteFiscal no quede bloqueada, devolviéndo el error –9, indicando que el impresor está ocupado. En éstas condiciones enviando el comando STATPRN, luego de que el problema del impresor desaparece, se obtiene la respuesta original al comando enviado. Esto facilita el manejo de los errores desde el programa.

- `int  SetBaudRate  (int PortDescriptor, long Baudios);`

Función para fijar la velocidad del puerto serie de la PC.

- `int  SearchPrn  (int PortDescriptor, long *Baud);`

Función que busca al controlador fiscal a las distintas velocidades posibles. Devuelve 0 si encontró la velocidad, almacenando en la variable Baud la velocidad a la que lo encontró. Si no lo encuentra, la función devuelve –1.

- `int  SetCommandRetries  (int Retries);`

Fija la cantidad de reintentos que efectúa la función MandaPaqueteFiscal para dar el paquete como enviado antes de devolver error.

- `int  SetNewProtocol  (int Value);`

Fija el uso del nuevo protocolo. No válido para impresores 614, 615, 951, PR4, 262.

- `void ObtenerNumeroDePaquetes  (int *Enviado, int *Recibido);`

Obtiene los números de paquetes que la librería envió y recibió en el comando.

- `int  ObtenerStatusImpresor  (int PortDescriptor, unsigned short *FiscalStatus, unsigned short *PrinterStatus, char *AnswerBuffer);`

Envía el comando STATPRN al impresor, devolviendo la respuesta de la misma forma que lo hace MandaPaqueteFiscal.

EJEMPLO DE USO DE LA LIBRERIA:

```
/*
testlib.c
Programa de prueba de la libreria fiscal.
Compilar y linkear con la libreria de la siguiente manera:

       $ cc -c testlib.c
       $ cc -o testlib testlib.o fislib.a -lx
*/

#include "fislib.h"
#include <unistd.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define FS                    0x1c
#define CMD_STATPRN 0xa1

static int CountDC4 = 1;
static int CountDC2 = 1;
static int fdtty;
static unsigned short PrinterSt;
static unsigned short FiscalSt;
static char Answer[500];

#ifdef DEBUG
void
FISdebug (char *fmt, ...)
{
       va_list argptr;
       static char BufLog [500];
       va_start(argptr, fmt);
       vsprintf (BufLog, fmt, argptr);
       va_end (argptr);
       printf ("DEBUG : [%s] \r\n", BufLog);
}
#endif

void
Show (char *Msg, char *buf)
{
       int c;

       printf ("%s", Msg);

       while ( *buf )
       {
              if ( (c = *buf++) == FS )
                     putchar ('|');
              else putchar ( c );
       }
}
int
ChequearImpresor (int ErrCode)
{
       int Rc;
       char buffer[120];

       // Si la funcion MandaPaqueteFiscal devuelve ERR_STATPRN,
       // es que el impresor no puede imprimir por falta de papel,
       // error mecanico o no esta OnLine.

       // Esta no es la respuesta al comando que se mando. Es una
       // respuesta de error en la que el usuario puede enviar
       // un mensaje al operador, indicando que solucione el problema
       // del impresor.

       // Esta logica es valida solo para el protocolo nuevo e impresores
       // nuevos, y evita que el programa quede bloqueado, esperando al
       // impresor que salga de su estado de error.

       // Una vez recuperado, el impresor manda la respuesta original

       do
       {
              printf (" *** Error de Impresor *** \r\n");

              sprintf (buffer, "%c", CMD_STATPRN);
```

```
                // Envia el comando CMD_STATPRN que encuesta al impresor.
                // Si este salio de su estado de error, envia la respuesta
                // al comando original.

                Rc = MandaPaqueteFiscal (fdtty, buffer, &FiscalSt,
                        &PrinterSt, Answer);

                if ( Rc != ERR_STATPRN && Rc != 0 )
                {
                        printf ("Error de comunicacion con el impresor \r\n");
                        exit (1);
                }

                sleep (2);
        }
        while ( Rc == ERR_STATPRN );

        return 0;
}

// Si el impresor detecta que falta papel, internamente se queda enviando
// el caracter DC4 por la linea serie hasta que el problema queda
// solucionado. Por otro lado si el impresor esta ocupado procesando
// algun comando envia el caracter DC2.
// Esta funcion es ejecutada por la libreria toda vez que el impresor
// envia los caracteres DC2 y DC4.
// Los contadores CountDC2 y CountDC4 deben ser puestos en 1 para cada
// comando que se envie al impresor.

void
KeepAlive (int Caracter, int Port)
{
        if ( Caracter == DC4 && !(++CountDC4 % 10))
                printf ("Falta Papel ... \r\n");

        if ( Caracter == DC2 && !(++CountDC2 % 30) )
                printf ("Impresor Ocupado... \r\n");
}

void
ResetCounters (void)
{
        CountDC4 = 1;
        CountDC2 = 1;
}

int
main (int argc, char *argv[])
{
        int i;
        FILE *fp;
        char buffer[120];
        int Rc;
        long BaudsPrn;
        int FlagNewProtocol = 1;

        if (argc != 2 )
                exit (1);

        #ifdef LINUX
        fdtty = OpenCommFiscal ("ttyS0");
        #else
        fdtty = OpenCommFiscal ("tty1a");
        #endif

        if ( fdtty < 0 )
        {
                printf ("Error abriendo el port\r\n");
                exit (1);
        }

        if ( FlagNewProtocol )
        {
                // Seteo de velocidad y nuevo protocolo, con manejo de
                // STATPRN.
                // No valido para impresoras PR4, 615, 614, 951, 262
                // (comentar
                // estas lineas si se trata de un impresor de los
                // mencionados).
                // En caso de no saber la velocidad del controlador,
                // SearchPrn
```

```
                // debe ejecutarse a continuacion del OpenCommFiscal, con el
                // objetivo de fijar la velocidad de comunicacion.

                printf ("Fijando modo de trabajo en protocolo nuevo ..\r\n");

                SetNewProtocol (1);

                printf ("Buscando controlador fiscal ..\r\n");

                if ( SearchPrn (fdtty, &BaudsPrn) < 0 )
                {
                        printf ("Error tratando de fijar la velocidad\r\n");
                        exit (1);
                }

                printf ("Controlador detectado a %ld\r\n", BaudsPrn);
        }

        else SetKeepAliveHandler (KeepAlive);

        InitFiscal (fdtty);

        if ( (fp = fopen (argv[1], "r")) == NULL )
        {
                printf ("Error abriendo %s\r\n", argv[1]);
                exit (1);
        }

        while ( fgets (buffer, sizeof (buffer), fp) )
        {
                Show ("Envio: ", buffer);

                // Reset de los contadores de DC2 y DC4.
                ResetCounters ();

                Rc = MandaPaqueteFiscal (fdtty, buffer, &FiscalSt,
                        &PrinterSt, Answer);

                if ( Rc < 0 )
                {
                        if ( FlagNewProtocol && Rc == ERR_STATPRN )
                                ChequearImpresor (Rc);

                        else
                        {
                                printf ("Error de comunicacion Rc = %d\r\n",
                                Rc);
                                exit (1);
                        }
                }

                printf ("FiscalStatus:  %04X\r\n", FiscalSt);
                printf ("PrinterStatus: %04X\r\n", PrinterSt);

                Show ("Answer:        ", Answer);

                printf ("\r\n\n");
        }

        fclose (fp);

        CloseCommFiscal (fdtty);
}
```

Apéndice A: Tratamiento de la respuesta fiscal.

Para recuperar el estado fiscal y el estado del impresor, se debe recurrir a separar en campos la respuesta y examinar sus bits. En Windows, estos valores son devueltos en cada operación en variables enteras, por lo cual no es necesario "rescatarlas" del string de la respuesta. Con los drivers de DOS no hay otra forma. Para hacer esto, veamos un ejemplo en lenguaje Basic:

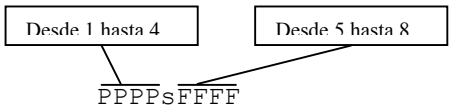
El string de respuesta siempre comienza de la siguiente forma:

"PPPPsFFFF"

donde:

PPPP: Estado del impresor (cuatro dígitos hexadecimales)
s: Separador de campos (ASCII 28).
FFFF: Estado fiscal (cuatro dígitos hexadecimales).

como sabemos que estas posiciones son fijas, las deducimos de la siguiente forma:



para lo cual usamos la función `Left$()` para los primeros cuatro caracteres, y `Mid$()` para los cuatro caracteres que siguen al caracter número 5.

```
'... luego de obtener la respuesta en la variable Respuesta...

Dim PrinterStatus as String
Dim FiscalStatus as String

PrinterStatus = Left$(Respuesta, 4)
FiscalStatus = Mid$(Respuesta, 5, 4)
```

Con esto separamos en dos variables de tipo string el estado fiscal y la estado del impresor.
Si se hubiera necesitado un campo de longitud variable (como podría ser cualquier otro campo adicional de la respuesta), habría que haber usado la función `InStr()` para buscar el separador de campos tantas veces como campos hay que saltar para determinar el comienzo del campo buscado. `InStr()` toma como primer parámetro opcional la posición a partir de la cual buscar, por lo que en las sucesivas llamadas hay que pasar la posición encontrada anterior más uno:

```
Dim n as Integer, m as Integer
Dim PrimerCampo as String, SegundoCampo as String

n = InStr(Respuesta, chr$(28))
m = InStr(n + 1, Respuesta, chr$(28))

PrimerCampo = Left$(Respuesta,n - 1)
SegundoCampo = Mid$(Respuesta, n + 1, m - n - 1)
```

Una vez extraídos los substrings, se presenta otro problema: estos dos valores están en formato hexadecimal, y hay que convertirlos a variables enteras para poder usarlos:

```
'...asumiendo que en PrinterStatus y en FiscalStatus están los
'   substrings recién extractados...

Dim Printer as Integer
Dim Fiscal as Integer

Printer = Val("&H" + PrinterStatus)
Fiscal = Val("&H" + FiscalStatus)
```

El string "&H" fuerza a la función Val (que convierte un string a un entero) a considerar el string como hexadecimal. Una vez hecho esto, ya estamos preparados para examinar los bits:

```
'... en la variable Printer está el status del printer como integer...

Const ERRORPRINTER = 4

if (Printer And ERRORPRINTER) = ERRORPRINTER then
      Print "El bit de error en printer está prendido"
end if
```

Este ejemplo examina el bit de error de printer dentro de la palabra de estado del printer mediante una operación "AND" a nivel de bits.
Por ejemplo, si el estado del printer era C084, la comparación efectuada es la siguiente:

```
      C084        1100000010000100
            AND
      0004        0000000000000100
             =
      0004        0000000000000100
```

Cualquier bit que en la máscara elegida (en este caso 4) esté en cero, queda en cero en el resultado, y cualquiera que estuviera en uno, queda en uno si en el número testeado ya estaba en uno. De forma que si el resultado es distinto que cero, alguno de los bits prendidos de la máscara estaban también prendidos en el número chequeado. Si el resultado es exactamente el de la máscara, exactamente todos los bits de la máscara estaban prendidos. Este tipo de operaciones con máscaras pueden ser combinadas:

```
'... en la variable Fiscal está el status fiscal como integer...

Const FISCALMEMFAIL = &H1
Const WORKINGMEMFAIL = &H2
Const FISCALMEMFULL = &H80

Dim ErrorCritico AS Integer

ErrorCritico = FISCALMEMFULL OR WORKINGMEMFAIL OR FISCALMEMFAIL

if (Fiscal and ErrorCritico) <> 0 Then
      Print "Hay un error fiscal crítico"
end if
```

Otra forma de hacerlo es mediante un array de estados:

```
'... suponiendo que en la variable entera FiscalStatus se encuentra el
'    estado fiscal devuelto por el impresor...

Dim Mask As Long
Dim i As Integer
Dim FiscalStatusError (16) as String

FiscalStatusError (1)  = "Fiscal Memory Fail"
FiscalStatusError (2)  = "Working Memory Fail"
FiscalStatusError (3)  = ""
FiscalStatusError (4)  = "Unrecognized Command"
FiscalStatusError (5)  = "Invalid Field Data"
FiscalStatusError (6)  = "Invalid Command"
FiscalStatusError (7)  = "Total Overflow"
FiscalStatusError (8)  = "Fiscal Memory Full"
FiscalStatusError (9)  = "Fiscal Memory Near Full"
FiscalStatusError (10) = "Fiscal Terminal Certified"
FiscalStatusError (11) = "Fiscal Terminal Fiscalized"
FiscalStatusError (12) = "Date Set Fail"
FiscalStatusError (13) = "Fiscal Receipt Opened"
FiscalStatusError (14) = "Receipt Opened"
FiscalStatusError (15) = "Invoice Opened"
FiscalStatusError (16) = ""

Mask = 1

For i = 1 To 16
      if (FiscalStatus And Mask) <> 0 then
            Print FiscalStatusError(i)
      end if
      Mask = Mask * 2
Next i
```

Aquí lo que se hace es prender uno a uno los bits para testearlos contra nuestro status fiscal. *Mask* vale uno en primera instancia, y se va multiplicando por dos en cada iteración, produciendo cada una de las potencias de dos:

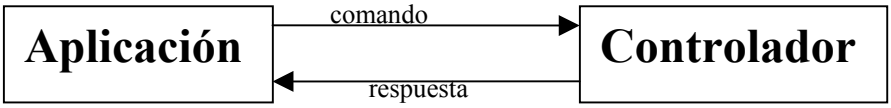
1	0001
2	0010
4	0100
8	1000

La variable *Mask* tiene que ser de tipo *long* porque de otra manera provoca un overflow al sobrepasar la frontera del entero con signo (32767).

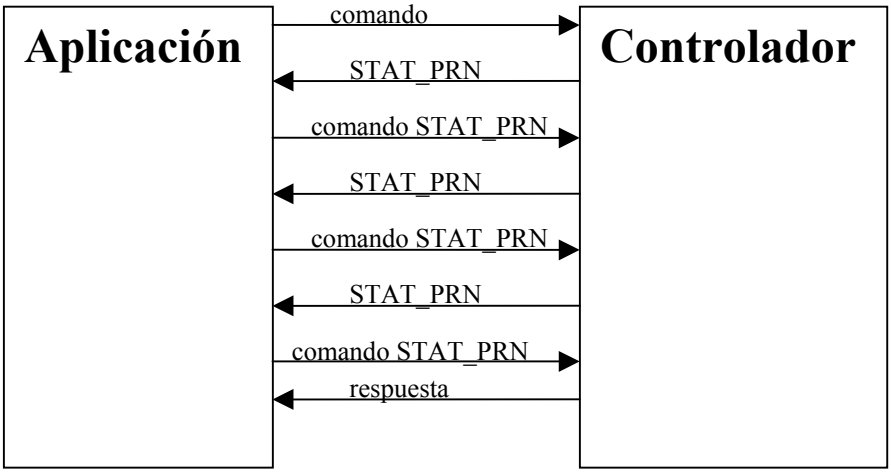
Apéndice B: Uso del modo STAT_PRN.

En los modelos más recientes de impresores, se dispone de una modalidad de trabajo que permite enterarse de errores mecánicos de impresor (enredo de papel, falta de papel, etc.) durante la ejecución de comandos, cosa que antes era imposible. El protocolo ha de activarse de alguna forma (consultar la documentación de los respectivos drivers a usar) y funciona de la siguiente forma:

Caso Normal:



Caso falta de papel:



Durante el estado de falta de papel, el controlador enviará a través del driver respuestas especiales del tipo STAT_PRN para que la aplicación pueda avisar al usuario de los problemas.
Si la aplicación detecta una respuesta del tipo STAT_PRN ante un comando, puede encuestar al controlador con comandos especiales (STAT_PRN) para saber si el controlador sigue en el estado de error. Cuando el controlador salga de ese estado, devolverá la respuesta correspondiente al comando que generó el error. En cada driver, se podrá determinar si la respuesta recibida es especial (STAT_PRN) o no. El siguiente es un algoritmo posible, escrito en pseudocódigo:

```
Enviar (Comando)
si Respuesta = STAT_PRN entonces
      Hacer
            AvisarUsuario (Respuesta)
            Enviar (STAT_PRN)
      Hasta que Respuesta <> STAT_PRN
Fin Si
Analizar (Respuesta)
```

donde la función `Enviar()` envía un comando al controlador, `AvisarUsuario()` avisa por pantalla a partir de la respuesta qué está andando mal, y `Analizar()` analiza la respuesta del comando.

U

V

W