OK CASH ECR-PC INTERFACE

THE PC COMMUNICATION DRIVER

Introduction

The Electronic Cash Register (ECR) model OK CASH is equipped with two RS-232 serial ports: the first one for the connection to an external computer, typically a Personal Computer (PC); the second one to connect a Bar Code Reader.

Through the communication between ECR and PC, it is possible to improve the ECR functions running applications on the PC interactively with the ECR. Examples of such applications are:

- management of a data base of articles in order to perform the sales using the Bar Code Reader;
- monitoring of the activity performed on the ECR to implement the Data Collection or to interact introducing additional functions (like special discount management);
- ECR data customisation, like PLU price and description updating.

In the protocol between the ECR and the PC there are exact rules and timings that, if not respected, can generate communication error.

The driver ECR232A8.SYS was especially designed to implement this protocol in a MS-DOS Device Driver so that applications running on the PC can exchange data with the ECR in a very easy way. It is able to handle up eight independent RS232 ports, either the standard COM1 and COM2 either non standard serial expansion boards.

Installation

To install the communication driver on the PC, just copy the file ECR232A8.SYS on the PC disk. It is not needed for the directory containing it to be inserted in the Path.

To make the operating system MS-DOS to load and activate the driver, add in the file CONFIG.SYS the following command line:

DEVICE=\dirname\ECR232A8.SYS

where "dirname" is the name of the directory where ECR232A8.SYS has been copied into.

In this case, by default, the driver will initialise COM1 and COM2.

When using ports different from COM1 and COM2 or using additional non standard ports, it is possible to customise the ports to be used by adding as option in the command line the base address and the interrupt of each RS232 port (up to eight):

DEVICE=\nomedir\ECR232A8.SYS /AAAA /I /AAAA /I /AAAA /I /AAAA /I

where:

AAAA	must be a 4 digit hexadecimal value specifying the port address of each port.
I	must be the number of the interrupt used by each port.

Legal interrupt numbers are 3, 4, 5, 6 and 7. More then one port can share the same interrupt (check the hardware configuration of the expansion board).

The hardware definition for the standard ports is the following:

PORT	I/O base address	Interrupt
COM1	03F8h	4
COM2	02F8h	3

If needed, it is also possible to exclude one of the standard ports, for instance because it is already used for a different purpose (i.e. the port COM1 is connected to a mouse or a modem), by specifying a zero address in the options added to previous described command line; i.e. the command line:

DEVICE=\nomedir\ECR232A8.SYS /0000 /0

will activate the driver only on COM2, while the command line:

DEVICE=\nomedir\ECR232A8.SYS /03F8 /4 /0000 /0

will activate the driver only on COM1, excluding COM2.

The default communication parameters are:

- 9600 bit/sec
- 8 bit data length
- No parity
- 2 stop bit

In a following paragraph will be explained how it is possible to select different communication parameters from the user program.

Further details on the CONFIG.SYS files are contained in the MS- DOS user manual.

How to Use the Driver

The MS-DOS identifies the driver with the names "OK232CH1", "OK232CH2", "OK232CH3", "OK232CH4" and "OK232CHX". For compatibility with the previous release of the driver, the first four names are referring respectively to the first four RS232 ports handled by the driver. When an user program opens a file using "OK232CH2" as name, the MS-DOS will re-direct to the second RS232 port all the following Read and Write operation performed on this file.

The name "OK232CHX" can be used to address any of the eight ports handled by the driver. In this case, the RS232 port number is specified in the data structure exchanged between the applicative program and the driver.

Driver Opening & Closing Procedure

To exchange messages with an ECR using the Driver is enough to open a "file of bytes" (or a "file of characters") with name "OK232CHi" (were i stands for 1, 2, 3, 4 or X) and make Read or Write operations, following the rules explained in the following paragraphs.

EXAMPLE : DRIVER OPENING/CLOSING PROCEDURE IN PASCAL LANGUAGE:

```
var  Ecr : file of byte;
begin
  assign (Ecr,'OK232CH1');
  reset  (Ecr);
     .
     .
     .
  close  (Ecr);
end.
```

Reception and Transmission Buffer

To exchange data and commands, the driver expects a buffer structured as follows:

```
Code      : byte                (or character)
PortNumber: byte                (or character)
Length    : integer (two bytes long)
Data      : array of characters  (or bytes)
```

The previous buffer must be defined as a contiguous structure, maintaining the described order for the variables. The meaning of the fields is:

Field	Meaning
Code:	Command Code / Error Code. Before calling the Read or Write operation, the user program must initialise it with the code that identifies the options requested while reading or writing. After the Read or the Write, the driver will fill this field with the Error Code of the operation.
PortNumber	This field has meaning only for Read and Write directed to "OK232CHX". A value between 1 and 8 will tell the driver to address the Read and Write operation to a specific RS232 port. A value 0 during a Read operation will ask the driver to check all eight ports for data. The value returned in this field by successful Read is the number of the RS232 port from which the data have been received. Calling a Write specifying 0 in this field, will ask the driver to send the data from the same RS232 port of the last successful Read.
Length	Length of the Data field. Before the Write operation, the user programs must initialise it with the number of characters to be sent to the ECR. Before the Read operation, the user programs must initialise it with the max length available for the Data field. After a successful Read this field contains the number of characters received from the ECR.
Data	Area that contains the character to be sent to or received from the ECR. The dimension of this field (in terms of number of characters) must be greater than the longest message to be exchanged with the ECR.

EXAMPLE : DEFINITION OF THE TX AND RX BUFFERS USING THE PASCAL:

```
type  ComBuff = record
                  Code      : byte;
                  PortNumber: byte;
                  Length    : integer;
                  Data      : array [1 .. 200] of char;
                end;

var  RxBuff : ComBuff;
     TxBuff : ComBuff;
```

- ***Note: The data field is defined as an array of characters, instead as a String, to make the driver be independent from the language used in the user program development. Different languages can have different internal representation for string type variables.***

Read & Write

To be compliant to the protocol's rules, the Driver performs the complete transmission or reception operation in a single MS-DOS activation.

To do so, the user program must call the Read or Write as if only the Code byte should be read or written. The driver will access autonomously the rest of the structure of the buffer.

READ AND WRITE SYNTAX TO INTERFACE THE DRIVER USING THE PASCAL LANGUAGE:

```
read (Ecr, RxBuff.Code);
write(Ecr, TxBuff.Code);
```

- ***Note: It is very important that the "Ecr" file is defined accordingly to the Code field definition ("file of byte" if the Code field is a byte; "file of char" if the Code field is a char) so that no "formatting" operations are performed within the Read or Write execution.***

To better clarify this concept with an example, if "Ecr" is defined as a text file, before writing the Code byte on the text file, the Write procedure will transform the byte content (that is numerical) in its text representation (that is ASCII), passing this to the driver instead of the defined data structure. Or also, using in Pascal the statement "writeln" instead of "write", would cause the queuing of Carriage Return and Line Feed characters to the Code byte. In both cases, the driver would not work.

Value of the Code Field

Following are listed the possible value of the Code fields with their meaning.

VALID CODES BEFORE THE WRITE:

Code	Description
0	requests the driver to transmit the Data field to the ECR.
3	It is not a transmission request. It is a command directed to the driver to activate the "auto answer mode". Nothing is transmitted to the ECR. When this mode is active, the driver perform by itself the handshake with the ECR so that no time-out occurs while executing on the PC programs that doesn't interface the ECR. Any Read or Write operation different from this, will disable the "auto answer mode".
5	It is not a transmission request. It is a command directed to the driver to initialise and re configure the communication port, clearing its internal buffer. Nothing is transmitted to the ECR. The buffer must contain the communication parameters, as follows:

```
Code       = 5 (initialise command)
PortNumber = 1÷8 (RS232 port to be initialised)
Length     = 5 (number of characters in the Data field)
Data[1]    = not used
Data[2]    = baud rate: '4' -  1200
                        '6' -  4800
                        '7' -  9600
Data[3]    = parity : 'N' - No Parity
Data[4]    = stop bits : '2'
Data[8]    = bit per character: '8'
```

- ***Note: The baud rate must match with the one selected in the ECR (default 9600). The last three parameter must be 'N', '2' and '8'.***

VALID CODES BEFORE THE READ

Code	Description
0	requests the driver to receive a buffer from the ECR. In this case, the Read will return only when a complete message is received from the ECR.

It is possible, setting one or more option bits, to obtain different functions:

Option	Meaning
10 hex	forces the Read to return also when a "Waiting For Keyboard Entry" message has been received from the ECR. If this occurs, after the Read the buffer will have both Code field and Length field set to 0. This is useful to synchronise PC and ECR when sending keyboard command from PC.
20 hex	forces the Read to return also when no message has been received from the ECR. If this occurs, after the Read the buffer will have the Code field set to 2.
40 hex	forces the Read to return also when a key is entered from the PC keyboard. If this occurs, after the Read the buffer will have the Code field set to 8.
80 hex	Forces the Read to return also when just numeric keys have been receives. Otherwise, the numeric keys are queued to build sequence terminated by at least one functional key.

VALID CODES AFTER THE READ OR THE WRITE

Code	Description
0	message successfully received (read) or transmitted (write) or Command executed (write).
1	error during transmission.
2	receive buffer empty. It works in conjunction with the 20 hex option bit in the Read code.
4	Data field too short. The message received is longer then the Data field length (one specified in the Length field before the Read call). The message is lost.
7	Time Out. It happens when an initiated communication session is not completed within the stated time.
8	Key Pressed on the PC keyboard. It works in conjunction with the 20 hex option bit in the Read code.
9	During the Memory Dump function, if the buffer is not large enough to contain all the data, only part of the data are passed. This code indicates that the Memory Dump is still in progress and a further Read is needed. When the transfer is completed, the code 0 will be returned.

- ***Note: The values described for the code field are numerical. That means that when the Code field is defined as a character, care must be taken while initialising it.***

Following is an example, in Pascal, of the right way to initialise the driver:

```
type  ComBuff = record
                  Code    : char;
                  PortNumber: char;
                  Length  : integer;
                  Data    : array [1 .. 200] of char;
                end;

var  TxBuff : ComBuff;
     Ecr    : file of char;
begin
  assign(Ecr,'OK232CH1');
  reset(Ecr);

  TxBuff.Code   := chr(5); {puts a numerical value into a character}
  TxBuff.Length := 5;
  TxBuff.data[1]:= 0;
  TxBuff.data[2]:= '7';
  TxBuff.data[3]:= 'N';
  TxBuff.data[4]:= '2';
  TxBuff.data[5]:= '8';
  write(Ecr, TxBuff.Code);
```

Writing

```
TxBuff.Code := '5';
```

instead of

```
TxBuff.Code := chr(5);
```

would initialise the Code field with the 53 (that is the value for the ASCII representation for the character '5').

Program Examples

Following are two examples of programs written in Pascal language.

EXAMPLE 1

The following program receives and shows on the computer screen all the messages sent by the ECR's:

```
{ Rx and Tx buffers definition }

type  ComBuff = record
                  Code      : byte;
                  PortNumber: byte;
                  Length    : integer;
                  Data      : array [1 .. 200] of char;
                end;

var  RxBuff : ComBuff;
     TxBuff : ComBuff;


{ constant code definition }

const  CodeOK       =   0 ;
       NoRX         =   2 ;
       PcKeyPressed =   8 ;
       Initialise   =   5 ;
       EcrSlave     =  $10;
       NoWait       =  $20;
       PcKeyCtrl    =  $40;


{ variables definition }

var  i      : integer;
     ch     : char;
     Ecr    : file of byte;


{ program entry point }

begin

{ driver opening }

  assign(Ecr,'OK232CHX');
  reset(Ecr);
```

```
{ configuration of the 8 serial ports }
  for i=1 to 8 do begin
    TxBuff.Code:=Initialise;
    TxBuff.PortNumber:=i;
    TxBuff.Length:=5;
    TxBuff.data[1]:= 0;
    TxBuff.Data[2]:='7';
    TxBuff.Data[3]:='N';
    TxBuff.Data[4]:='2';
    TxBuff.Data[5]:='8';
    write(Ecr, TxBuff.Code);
  end;

  repeat

{ reception of a message from ECR's with option of return if PC key pressed
}
    RxBuff.Code:=PcKeyCtrl;
    TxBuff.PortNumber:=0;
    RxBuff.Length:=200;
    read (Ecr,RxBuff.Code);

{ if a message was received, its content is shown }
    if (RxBuff.Code=CodeOk)
       then begin
         writeln('Received data from ECR ', TxBuff.PortNumber);
         for i:=1 to RxBuff.Length do write(RxBuff.Data[i]);
         writeln;
       end

{ if an error code condition is detected it is signalled }
       else if (RxBuff.Code<>PcKeyPressed) then
                 writeln('    DRIVER ERROR ', RxBuff.Code);

{ if a key is pressed on the PC keyboard, the program terminates }
  until RxBuff.Code=PcKeyPressed;

  close(Ecr);
end.
```

EXAMPLE 2

The following program reads a keyboard command message from keyboard and sends it to the ECR.

```
{ Rx and Tx buffers definition }

type  ComBuff = record
                  Code      : byte;
                  PortNumber: byte;
                  Length    : integer;
                  Data      : array [1 .. 200] of char;
                end;

var  RxBuff : ComBuff;
     TxBuff : ComBuff;


{ constant code definition }

const  CodeOK       =   0 ;
       NoRX         =   2 ;
       PcKeyPressed =   8 ;
       Initialise   =   5 ;
       AutoAnswer   =   3 ;
       EcrSlave     =  $10;
       NoWait       =  $20;
       PcKeyCtrl    =  $40;


{ variables definition }

var  i:integer;
     ch : char;
     Command : string;
     Ecr : file of byte;


{ program entry point }

begin
{ driver opening }
  assign(Ecr,'OK232CH1');
  reset(Ecr);

{ Selection and reconfiguration of the serial port }
  writeln('Serial port to be used (1..4)?');
  readln(ch);
{ configuration of the 8 serial ports }
```

```
for i=1 to 8 do begin
  TxBuff.Code:=Initialise;
  TxBuff.PortNumber:=i;
  TxBuff.Length:=5;
  TxBuff.Data[1]:=0;
  TxBuff.Data[2]:='7';
  TxBuff.Data[3]:='N';
  TxBuff.Data[4]:='2';
  TxBuff.Data[5]:='8';
  write(Ecr, TxBuff.Code);
end;

repeat

   writeln('Command to be sent ? (RETURN to quit)');

{ Waiting the keyboard entry from the PC the program shows }
{ the activity on going on the ECR. This loop ends at the  }
{ first key entry on the PC                                }

   repeat
       RxBuff.Code:=PcKeyCtrl;
       TxBuff.PortNumber:=0;
       RxBuff.Length:=200;
       read (Ecr,RxBuff.Code);
       if (RxBuff.Code=0)
          then begin
            writeln('Received data from ECR ', TxBuff.PortNumber);
            for i:=1 to RxBuff.Length do write (RxBuff.Data[i]);
            writeln;
            writeln('Command to be sent ? (RETURN to quit)');
          end;
    until RxBuff.Code=PcKeyPressed;

{ A key pressed on the PC ends the monitor loops              }
{ before reading the desired command from the PC keyboard,    }
{ it is activated the Auto Answer on the driver, so that      }
{ the ECR doesn't disconnects itself during the PC data reading }

  for i:=1 to 8 do begin
    TxBuff.Code:=AutoAnswer;
    TxBuff.PortNumber:=i;
    write(Ecr, TxBuff.Code);
  end;


{ the command line is read from PC keyboard }
```

```
    readln(Command);

    if (length(Command) > 0) then
       begin
{ Selection of the serial port }
         writeln('Serial port to be used (1..8)?');
         readln(RxBuff.PortNumber);

{ preparing the transmission buffer      }
         TxBuff.Code:=0;
         TxBuff.Length:=length(Command);
         TxBuff.PortNumber:=0;
         for i:=1 to length(Command) do TxBuff.Data[i]:=Command[i];

{ waits for the ECR to be ready to accept a remote keyboard entry }

         RxBuff.Code:=PcKeyCtrl+EcrSlave;
         RxBuff.Length:=200;
         read (Ecr,RxBuff.Code);

{ if the ECR can accept the command, sends the buffer      }
         if (RxBuff.Code=CodeOk) and (RxBuff.Length=0)
             then write(Ecr,TxBuff.Code);

{ if a key was pressed on the PC while waiting, the program stops }
         if (RxBuff.Code=PcKeyPressed) then Command:='';
       end;

  until (length(Command)=0);

  close(Ecr);
end.
```

LIBRARY FOR WINDOWS 95

The Dynamic-Link Library **OkInterf.dll** has been developed to simplify the connection between a PC and the OK Cash ECR's using programs operating under Windows 95. The **OkInterf.dll** is a 32 bits library, therefore can be used only in conjunction to 32bits compilers.

This library has the capability to handle up to a maximum of 8 independent RS232 ports, either the standard COM1 e COM2, either non standard ports installed in serial expansion boards, but only if a Windows Communication Devices is available for these boards to operate under Windows 95.

It is important that the program implementing the ECR's interface management performs cyclically the call to the receiving procedure, otherwise a communication time-out will occur on the ECR's side.

The available library procedures are described thereafter. The operative mode is similar to that described for the MS-DOS driver.

EcrOpen

The procedure definition is as follows:

void EcrOpen (char *Porta, int *EcrNum, int BaudRate)

Must be called to open a Windows Communication Device and assign a Cash Register Number to a serial port.

Calling Parameters:

char ***Porta	Pointer to a null terminated string containing the name of the Communication Device to be opened (i.e.: "COM1", "COM2").
int *EcrNum	Pointer to a 32 bits integer variable, containing the ECR number to be associated to the Communication Device. Valid values for ECR numbers are 1 to 8 or 0. A value 0 will result in an automatic association of the first not assigned ECR number.
int BaudRate	A 32 bits integer variable, containing the value of the Baud Rate. Valid values are: 1200, 4800, 9600. Any invalid value will be treated as 9600 Baud. The remaining communication parameters are automatically set in accordance to the ECR protocol.

Returned Values:

EcrNum	After successful completion of the EcrOpen procedure, the variable will contain the ECR number linked to the serial port. A returned value -1 will indicate an failed open operation.

Possible causes of fail are:

- the Communication Device does not exists
- the Communication Device is already opened by other application
- all possible 8 ECR numbers are already assigned

EcrWrite

The procedure definition is as follows:

void EcrWrite (int *Codice, int *EcrNum, int Quanti, char *TxBuff)

It sends a data frame on the serial port to the specified ECR.

Calling Parameters:

int *Codice	Pointer to a 32 bits integer variable. The initial value of this parameter is ignored.
int *EcrNum	Pointer to a 32 bits integer variable, containing the ECR number to be associated to the Communication Device. Valid values for ECR numbers are 1 to 8. A value 0 will result in an automatic transmission to the last ECR from which a frame was received.
int Quanti	A 32 bits integer variable containing the numbers of characters to be transmitted.
char *TxBuff	Pointer to a string containing the caracters to be transmitted. The string is treated as an array of characters.

Returned Values:

Codice	After the EcrWrite procedure, the variable Codice will contain a code with the result of the write operation: • 0 = successful write • 1 = failed write operation
EcrNum	After the EcrWrite procedure, the variable will contain the contacted ECR number.

EcrRead

The procedure definition is as follows:

void EcrRead (int *Codice, int *EcrNum, int *Quanti, char *RxBuff)

It receives a data frame via serial port from the specified ECR.

Calling Parameters:

int *Codice	Pointer to a 32 bits integer variable. The initial value of this parameter shall contain the reading options. A valid value is the combination (sum) of the following options: 0 = normal read: the procedure will return when a valid record has been received, excluding "Waiting For Keyboard Entry" messages. 16 = Ecr keyboard control option: the read will return also when a "Waiting For Keyboard Entry" message is received from the ECR. In such a case, both Codice and Quanti will contain zero. This option is needed to synchronize the PC and the ECR sending commands from the PC; 32 = Not locking read option: the read will return even if no data have been received. In such a case, the value returned into Codice is 2; 128 = single key read option: the read will not pack the received keys to build a terminated keyboard string, but will return each single received keyboard command.
int *EcrNum	Pointer to a 32 bits integer variable, containing the ECR number to be tested for reception. Valid values for ECR numbers are 1 to 8 or 0. A value 0 will test all the opened ECR's for the first available received message. When a message is received, this variable will contain the number of the sender ECR.
int *Quanti	Pointer to a 32 bits integer variable containing the maximum numbers of characters that can be received (dimensions of the receive buffer RxBuff).
char *RxBuff	Pointer to a characters array to contain the received caracters.

Returned Values:

Codice	After the EcrRead procedure, the variable Codice will contain a code with the result of the read operation: 0 = read completed (data are in RxBuff) 1 = error during read operation 2 = no caracters have been received 4 = the received keyboard string is longer then RxBuffer (data are lost) 7 = time out 9 = a memory dump is still in progress (data received up to now are in RxBuff)
EcrNum	Number of the ECR whose data have been received (only when Codice = 0 or 9).
Quanti	Number of characters that have been received (only when Codice = 0 o 9).
RxBuff	Received data (only when Codice = 0 o 9).

EcrClose

The procedure definition is as follows:

void EcrClose (int EcrNum)

Must be called at the end of the program to release the Windows Communication Device previously assigned to a Cash Register Number.

If a Windows Communication Device is not released at the end of the program, it will remain locked and no further access is possible unless Windows restarts.

Calling Parameters:

int EcrNum	A 32 bits integer variable, containing the ECR number associated to the Communication Device to be released. Valid values for ECR numbers are 1 to 8. A value 0 will automatically close all previously opened devices.

INTERFACING A PC TO OK CASH ECR

Message Structure

The interaction between the ECR and a PC is obtained exchanging messages that contain, mainly, ECR keyboard entries.

Each ECR keyboard entry is represented by a string of ASCII characters. Then each ECR key has an associated character, according to the following table:

ECR KEY	CHARACTER	HEX ASCII CODE
0	0	30
1	1	31
2	2	32
3	3	33
4	4	34
5	5	35
6	6	36
7	7	37
8	8	38
9	9	39
.(decimal point)	.	2E
X (multiply)	*	2A
%	%	25
PLU	I	49
DEPARTMENT 1	P	50
DEPARTMENT 2	Q	51
DEPARTMENT 3	R	52
DEPARTMENT 4	S	53
DEPARTMENT 5	T	54
DEPARTMENT 6	U	55
DEPARTMENT 7	V	56
DEPARTMENT 8	W	57
SUBTOTAL	=	3D
TOTAL	X	58
TENDER 2	Y	59
TENDER 3	Z	5A
+	+	2B
-	-	2D
PAPER FEED	A	41
CLEAR	K	4B
ITEM CORRECT	L	4C

In addition to the above keys, there are the following special symbols: a string delimitator, used to enclose ASCII strings (article description); a protocol related wait command; a communication error indicator:

SYMBOL	CHARACTER	HEX ASCII CODE
string delimitator	"	22
serial number and version-revision delimitator	!	21
memory dump identifier	&	26
printer buffer identifier	a	61
printer buffer identifier	b	62
printer buffer identifier	c	63
printer buffer identifier	d	64
printer buffer identifier	e	65
printer buffer identifier	f	66
printer buffer identifier	g	67
printer buffer identifier	h	68
ECR error identifier	E	45
wait	@	40
Communication Error	G	47

Excluding the memory dump operation, all the messages exchanged between the ECR and the PC are sequences of ASCII characters.

Message Sequences

The communication session is always started by the ECR that sends or the keys entered by the cashier or an empty buffer if the ECR is waiting for a key entry.

After sending the key sequence, the ECR doesn't processes it, waiting for the PC answer. If no answer is received before the time-out occurs, the ECR disconnects itself.

The answer received from the PC is the key sequence interpreted and executed on the ECR, instead the very last one entered by the cashier.

This allows the PC even to change the entry done by the cashier (i.e. to insert special discounts).

- ***Note: To avoid mixed sequences (keyboard entered digits mixed to PC commands) the PC must clear the ECR entry by starting the answer string with the K character, if needed.***

It is also important to store partial entries, like multiplication, because to perform a sale from the PC (i.e. to manage the Bar Code) a complete sale sequence, including the eventual multiplying quantity or even the Void key, MUST be prepared.

In the following example, the PC adds an automatic discount to a sale performed on the PLU 115:

frames received from ECR	2.5*115I
frame answered back from the PC	K2.5*115I10%-

It is also important to note that multiple sequences can be received in a single message. For instance, if a department key is repeated to perform several sales, it is probable to receive:

first sale on the department	1500R
three time repetition	RRR

If the PC program should only monitor the ECR activity without interfering, it must echo each message. This ECHO function is performed in the DOS Driver ECR232A8.SYS each time no Write is done between two consecutive Read operations. Therefore, using the Driver, the PC program can ignore this echo, performing only reading operations to monitor the activity on going and writing only when the PC should interact with the ECR.

Examples of messages are:

3X1500P	(sales of 3 items with price=1500 on the department 1)
=3.5%-	(3.5 % discount on the subtotal)

Special Messages

Following are listed the message's that are not ECR keyboard entries.

ERROR MESSAGES (SENT BY THE ECR)

When an error occurs on the ECR, this error condition is sent also to the PC. The message format is the letter E followed by the numerical code of the error. Examples of the error message are:

Error	Meaning
E14E	(function not enabled)
E10E	(wrong keyboard sequence)
G	Communication error (sent by the ECR)

When a communication error is detected by the ECR (time out or wrong message format) the ECR disconnects itself after sending the character G by itself.

ITEM DESCRIPTION (SENT BY THE PC)

The PC can send to the ECR a description to be used in the very next department sale. The message to be sent is the description in ASCII enclosed into double quote. Should the item description use characters other then numeric (from ‘0’ to ‘9’) or letters (from ‘A’ to ‘Z’), the ECR internal character code, not always according to the ASCII code, must be (see paragraph **Sending Item Descriptions**).

The description must never be longer than 11 characters.

Example is: "COCA COLA"

BAR CODE ARTICLE NUMBER (SENT BY THE ECR)

If a bar code reader is connected to the ECR, when a code is read it is sent to the PC as a string enclosed in double quote, containing all the characters received from the bar code reader.

Example of this message is: “A051111125137”

If the PC has an article data base can answer to this message with a sales entry containing the description (optional), the price and the department.

An example of possible answer is: "FLOPPY DISK"15000R

MEMORY DUMP (SENT BY THE ECR)

If a Memory Dump has been requested to the ECR (Departments data, PLU data, etc. – see section 6 of the Manual for the list of available dump), the image of the memory is sent to the PC in a single frame (max. length 16 Kbytes). The first character of such message is: &.

RECEIPT PRINTOUT ECHO (SENT BY THE ECR)

All the print command directed to the internal receipt printer are echoed on the serial line. Such messages are composed by the letter "a" (or "b" or "c" or "d" or "e" or "f" or "g") followed by the printed string. Its length is according to the number of characters of the printer. The letter "h" by itself identifies a Line Feed command.

WAIT COMMAND

When the ECR sends a message, it expects an answer within 1 second. Elapsed this time, a time-out occurs causing a communication error.

If it is needed a longer time to prepare the answer, it is possible to restart on the ECR the time-out counter by sending the character @ before the elapsing of the previous one. It is possible to repeat several times this process to make the ECR to wait longer time.

- ***Note: An application program should not care about the Wait command because the driver ECR232A8.SYS automatically sends it in order to make the time-out as long as 8 seconds.***

IDENTIFICATION MESSAGE

This message is sent automatically by the ECR any time the communication is enabled through the sequence 204 SBT.

The structure and the content of this message is as follows:

!nnnnnnnnnnnn vv-rrssssss!<CR>

where:

vv	two digits to identify the software version. Example: 00: Not fiscal ECR's-messages in English 01: Fiscal ECR's-messages in Italian 03: Not fiscal ECR's-messages in German 09: Fiscal ECR's-messages in Russian 10: Not fiscal ECR's-messages in Slovak
rr	two digits to identify the revision of the software;
nnnnnnnnn	serial number of the ECR, stored into the Fiscal Memory (empty string for not fiscal ECR);
sssss	ECR model.

Following are two examples, the first one for a fiscal ECR, the second one for a not fiscal ECR:

!EK 00000001 15-5151-05!

When the connection is active, it is possible for the PC to request this message by sending the keyboard command:

204==

WRITING PROGRAMS FOR ECR INTERFACING

Scope

The aim of this document is to provide hints and examples, in order to help the writing of PC programs able to control the OK CASH through the RS232 serial port. For detail on how to install and interface the communication driver, refer to the section ECR/PC Interface.

Development Languages

Both the MS-DOS communication driver and the Windows 95 Dinamic Link Library are designed in such a way that PC programs can perform receiving and transmitting operations, as well as issuing mode control commands just using standard disk file access procedures or library functions. Nevertheless, the rules to properly define and use the interface structures, needed to correctly interface the driver, are different depending on the programming language used to develop the program.

Following, there are examples that show the correct driver interface procedure for the most popular programming languages.

Program Example Using Basic under MS-DOS

```
REM   ****** Interface buffers definition ******

TYPE CommBuffer
   Comando AS INTEGER
   Lung AS INTEGER
   Dati AS STRING * 200
END TYPE

COMMON SHARED RxBuff AS CommBuffer
COMMON SHARED TxBuff AS CommBuffer
COMMON SHARED st AS STRING


REM   ****** Driver Opening ******
   OPEN "OK232CH1" FOR BINARY AS #1 LEN = 1


REM   ****** Serial Port initialization ******
REM   ****** 9600 Baud, No Parity, ******
REM   ****** 2 Stop Bits, 8 Bits per Character ******
   TxBuff.Comando = 5
   TxBuff.Lung = 5
   TxBuff.Dati = "17N28"
   PUT #1, , TxBuff.Comando


   DO

REM   ****** reads an ECR message  ******
REM   ****** 64 requests the driver to return on PC Key pressed ******
     RxBuff.Comando = 64
     RxBuff.Lung = 200
     GET #1, , RxBuff.Comando

REM   ****** If a message has been received,     ******
REM   ****** extracts and prints the data string ******
     IF (RxBuff.Comando = 0) THEN
       st = LEFT$(RxBuff.Dati, RxBuff.Lung)
       PRINT st
     END IF

   LOOP UNTIL INKEY$ = CHR$(27)

   CLOSE #1
```

Program Example Using C Language under MS-DOS

```
#include "string.h"
#include "io.h"
#include "fcntl.h"
#include "stdio.h"

/*
  ****** Interface buffer definition ******
*/

struct Buffer
      2-2-{
      2-2-  int Command;
      2-2-  int Lung;
          char Line[200+1];
      2-2-} RxBuffer;

int i,j;
int Ecr;

void main()
{
/*
  ****** Driver Opening ******
*/
  Ecr=open("OK232CH1",O_RDWR+O_BINARY);

/*
  ****** Serial Port initialization ******
  ****** 9600 Baud, No Parity, ******
  ****** 2 Stop Bits, 8 Bits per Character ******
*/
  RxBuffer.Command=5;
  strcpy(RxBuffer.Line,"17N28");
  RxBuffer.Lung=strlen(RxBuffer.Line);
  write(Ecr,&RxBuffer,1);

  do
  {
/*
  ****** reads an ECR message  ******
  ****** 0x40 requests the driver to return on PC Key pressed ******
*/
    RxBuffer.Command=0x40;
    RxBuffer.Lung=200;
    read(Ecr,&RxBuffer,1);
```

```
/*
  ****** If a message has been received,     ******
  ****** terminates and prints the data string ******
*/
    if (RxBuffer.Command==0)
    {
      RxBuffer.Line[RxBuffer.Lung]=0;
      printf("%s \n",RxBuffer.Line);
    }
  }

/*
  ****** RxBuffer.Command==8 indicates a PC key pressed ******
*/
  while (RxBuffer.Command != 8);
  close(Ecr);
}
```

Program Example Using Pascal under MS-DOS

```
{ Rx and Tx buffers definition }

type  ComBuff = record
                  Code    : byte;
                  Spare   : byte;
                  Length  : integer;
                  Data    : array [1 .. 200] of char;
                end;

var  RxBuff : ComBuff;
     TxBuff : ComBuff;


{ constant code definition }

const  CodeOK       =   0 ;
       NoRX         =   2 ;
       PcKeyPressed =   8 ;
       Initialize   =   5 ;
       EcrSlave     =  $10;
       NoWait       =  $20;
       PcKeyCtrl    =  $40;


{ variables definition }

var  i      : integer;
     ch     : char;
     st     : string;
     Ecr    : file of byte;


{ program entry point }

begin

{
  ****** Driver Opening ******
}
  assign(Ecr,'OK232CH1');
  reset(Ecr);

{
  ****** Serial Port initialization ******
```

```
  ****** 9600 Baud, No Parity, ******
  ****** 2 Stop Bits, 8 Bits per Character ******
}
  TxBuff.Code:=Initialize;
  TxBuff.Length:=5;
  TxBuff.Data[1]:='1';
  TxBuff.Data[2]:='7';
  TxBuff.Data[3]:='N';
  TxBuff.Data[4]:='2';
  TxBuff.Data[5]:='8';
  write(Ecr, TxBuff.Code);


  repeat

{
  ****** reads an ECR message  ******
  ****** requesting the driver to return on PC Key pressed ******
}
    RxBuff.Code:=PcKeyCtrl;
    RxBuff.Length:=200;
    read (Ecr,RxBuff.Code);

{
  ****** If a message has been received,     ******
  ****** builds and prints the data string ******
}
    if (RxBuff.Code=CodeOk)
       then begin
         st:='';
         for i:=1 to RxBuff.Length do st:=st+RxBuff.Data[i];
         writeln(st);
       end

{ if a key is pressed on the PC keyboard, the program terminates }
  until RxBuff.Code=PcKeyPressed;

  close(Ecr);
end.
```

Program Example Using Visual Basic for Windows95

```
'
'  Program example for Visual Basic 5.0
'
'
'  Definition of the Rx and TX Buffer type as a char array

Public Type RxTxBuff
     Linea As String * 100
End Type

' Variables Definition

Global Ecr As Long
Global RxCommand As Long
Global RxLength As Long
Global TxCommand As Long
Global TxLength As Long

Global TxBuf As RxTxBuff
Global RxBuf As RxTxBuff

Global Data As String
Global KeySequence As String



'  Syntax of library procedures :

Declare Sub EcrOpen Lib "OkInterf" (ByRef PortName As RxTxBuff, ByRef Ecr-
Num As Long, ByVal BaudRate As Long)
Declare Sub EcrRead Lib "OkInterf" (ByRef Code As Long, ByRef EcrNum As
Long, ByRef Quanti As Long, ByRef RxBuff As RxTxBuff)
Declare Sub EcrWrite Lib "OkInterf" (ByRef Code As Long, ByRef EcrNum As
Long, ByVal Quanti As Long, ByRef TxBuff As RxTxBuff)
Declare Sub EcrClose Lib "OkInterf" (ByVal EcrNum As Long)




Sub OpenFiles()
'
'  Assigns next free logic number to COM2 and puts it into the variable ECR
'  On Fail, abort the program after a warning.
'
```

```
   Ecr = 0
   st3 = "COM2"

   TxBuf.Linea = st3 & Chr(0)

   Call EcrOpen(IniBuf, Ecr, 9600)

   If Ecr < 0 Then
        st3 = st3 & " Device Error !"
        MsgBox st3
        Call EcrClose(0)
        End
   End If


End Sub




Sub Receive()
'
'  While waiting for data from the port, executes the events.
'  Upon reception, returns the received string into Data.
'

   RxCommand = 32          ' inizialization

   Do While Command <> 0

        DoEvents           ' to execute the focused Form

        RxCommand = 32     ' non blocking read option
        RxLength = 100

        Call EcrRead(RxCommand, Ecr, RxLength, RxBuf)

   Loop

   Data = Left(RxBuf.Linea, RxLength)

End Sub



Sub Send()
'
```

```
'  Sends the KeySequence string content.
'

     TxCommand = 0
     TxLength = Len(KeySequence)
     IniBuf.Linea = KeySequence

     Call EcrWrite(TxCommand, Ecr, TxLength, TxBuf)

End Sub



Sub CloseFiles(Cancel As Integer)
'
'  Closes all opened Comm Devices.
'
    Call EcrClose(0)
    End
End Sub



Private Sub Form_Unload(Cancel As Integer)
'
' Closes all opened Comm Devices.
'
    Call EcrClose(0)
    End
End Sub


'
' The applicative program starts by calling OpenFiles;
' works calling cyclically Receive;
' when it has to send a key sequence to the ECR calls Send;
' At the end must call CloseFiles.
' This call is repeated into the Unload of the main Form to close the Comm
Device
' in the event that the program is terminated by anexternal event
'
```

Program Example Using Visual C for Windows95

```
(*
   In the Link optiond the librery OkInterf.lib must be included.

   This program listen to the ECR keyboard sequences, adding a
   10% discount on PLU 100 sales.
*)


// Library functions definition

void EcrOpen (char *Port, int *EcrNum, int BaudRate);
void EcrWrite (int *Code, int *EcrNum, int Quanti, char *TxBuff);
void EcrRead (int *Code, int *EcrNum, int *Quanti, char *RxBuff);
void EcrClose (int EcrNum);

// Costants Definition

#define RXOK        0    // Ricezione completa
#define NORX        2    // Nessuna ricezione
#define NoWait     32    // bit di read senza attesa
#define KeybCtrl   16    // bit di read con controllo tastiera ECR

int WINAPI WinMain( HINSTANCE hInstance, // handle to current instance
                    HINSTANCE  hPrevInstance,   //  handle  to  previous  in-
stance
                    LPSTR lpCmdLine,  // pointer to command line
                    int nCmdShow  // show state of window


                  )
{
 char Buffer[80];
 char Command[80];
 int  Mode;
 int  NumChar;
 int  NumEcr;

// Assigns next free logic number to COM2
 NumEcr = 0;
 EcrOpen ("COM2", &NumEcr, 9600);

 if (NumEcr >= 0)
 do
 {
// waits for a keyboard string
 do
```

```
   {
     Mode   = NoWait;
     NumChar= 70;
     EcrRead (&Mode, &NumEcr, &NumChar, Buffer);
   }
   while (Mode == NORX);

// if a string has been received
   if (Mode == RXOK)
   {
// verifies if it is a PLU 100 sale
       if (strcmp(Buffer,"100I") ==0 )
       {
// if yes, waits for the ECR in idle state
        do
          {
            Mode   = NoWait + KeybCtrl;
            NumChar= 70;
            EcrRead (&Mode, &NumEcr, &NumChar, Buffer);
          }
          while ((Mode != RXOK) || (NumChar != 0))

// Sends the 10% discount sequence
          strcpy(Command, "10%-");
          Mode = 0;
         EcrWrite (&Mode, &NumEcr, strlen(Command), Command);
       }
    }
  }
  while ((Buffer[0]!='G') && (strcmp(Buffer,"205=")!=0));

// on ECR off line, the program terminates

  EcrClose(0);
  return(0);
}
```

MESSAGES SENT BY THE ECR

The communication session is always started by the ECR. Therefore, the program on the PC, after port initialization, shall read the port first and, only when a message is received such that requires inter-actively from PC, it shall send the proper reply message. The examples mentioned in the following paragraphs, as the example programs contained in the floppy, are written in Pascal, using the MS-DOS driver. It is easy for a programmer to translate them in a different language.

ECR Identification Message

The OK CASH ECR uses this message to start the communication session. This means that, when the communication session is not yet opened, the ECR Identification message is received at the first read operation on the PC. The ECR sends the Identification Message in the following cases:

- when a "204 STL STL" command sequence is given to the ECR trough keyboard or from serial port (only if out of transaction);
- automatically each second when the communication is not active and the ECR is showing the time of day on the display;
- cyclically whenever, during the communication session, no answer is received from the PC or a communication error occurred (On the ECR display the "DISCONNECT PC" message is shown).

No direct answer is needed to this message (the driver will answer with the handshake message on the following read). The PC can analyse the string to determine which ECR is connected.

Note that on the very first read operation (if the port setting command was issued to the driver while the ECR is sending this message) only part of the string might be received (the final part). In such a case, the program shall not misunderstand the type of string (i.e. verifying that the string terminates with a '!'). The connection is established anyway. If the complete ECR Identification is needed to initialise the PC program, its repetition can be requested issuing from PC the command message

204== (the equivalent of keyboard sequence 204 STL STL).

In this case, being the communication session already started, the whole string will be received.

Keyboard Messages

When the communication is active, each key (or group of keys) pressed on the ECR keyboard is sent to the PC. When the PC echoes the key sequence, this is executed.

This is the base of the inter-activity between the PC and the ECR, allowing the PC also to change the key sequence itself.

If the scope of the PC connection is to perform data collection and ECR activity monitor, no need exists to modify the received key sequences. In this case the PC program shall perform only reading operation, and the driver will take care of sending back the received key sequences to the ECR (the previously received sequence will automatically echoed by the driver on the following read operation, if in the meantime no write has been requested from the PC program).

If the scope of the PC program implies the complete inter-activity (i.e. item sales using the bar code reader), then it will be necessary for the program to directly control the echo of some key sequence. To do this it is important to know how sequences are received, in order to correctly analyse them and build the proper reply.

A basic sequence is composed by a string containing one function key, eventually preceded by one or more numeric keys.

An item entry is composed by one or more basic sequences.

To better clarify, following are examples of item entries:

COMPLETE ITEM ENTRY	BASIC SEQUENCES	FUNCTION ACTIVATED
Q	Q	Sale on department 2 using the preset price
1500P	1500P	Sale on department 1 with an amount of 1500
152I	152I	sale on PLU 152
I	I	repetition of the previous PLU sale
2.5*1000R	2.	sale of a quantity of 2.5
	5*	with unit price 1000 on
	1000R	department 3
-2.5*850L100I	-	void of a sale of a quantity
	2.	of 2.5
	5*	
	850L	with unit price 1000
	100I	on open PLU 100

Keyboard messages are sent by the ECR as soon as key codes are taken from the keyboard queue. This means that when the ECR is in idle state, a key is sent when it is pressed; if the ECR is working (i.e. pressing keys when printing) the keys are queued and sent all together as a single message.

In order to maintain interactivity, when receiving a keyboard message, the driver immediately echoes numeric keys before receiving the first function key, passing the string to the calling program only when at least one function key and the end of message has been received. As a consequence of that, when reading the driver the following events can occur:

1. A SINGLE COMPLETE BASIC SEQUENCE IS RECEIVED ON EACH READ

(this is the most frequent event); i.e.:

2.	when reading this 2 is already echoed
5*	when reading this 5 is already echoed
1000R	when reading this 1000 is already echoed
100I	when reading this 100 is already echoed

2 - MORE THEN ONE BASIC SEQUENCE OR ITEM ENTRY IS RECEIVED. I.E.:

2.5*1000R100IIII	(when reading this 2 is already echoed)

3 - ONE OR MORE COMPLETE BASIC SEQUENCES PLUS THE STARTING PART OF THE FOLLOWING IS RECEIVED ON A READ AND THE REMAINING PART IS RECEIVED ON THE FOLLOWING READ. i.e.:

2.5*10	(when reading this 2 is already echoed)
00R	(when reading this 1000 is already echoed)
100IIII	(when reading this 100 is already echoed)

When receiving messages like in the examples, the program that analyses them must recognise the following item entries:

2.5*1000R	sale of quantity 2.5 at 1000 unit price on department 3
100I	sale of quantity 1 on PLU 100
I	repetition of the previous PLU sale
I	repetition of the previous PLU sale
I	repetition of the previous PLU sale

If it is needed for the PC program to modify partially the received sequence, for instance to add a special discount of 150 value after three sold items on PLU 100, a new key sequence must be built and sent to the cash register. Building this new key sequence, it is needed to think first how should be the proper translation of the basic sequences in order to make the ECR correctly perform the item entries and the discount in the appropriate order:

2.5*1000R	sale of quantity 2.5 at 1000 unit price on department 3
100I	first sale on PLU 100
I	repetition of the previous PLU sale
I	repetition of the previous PLU sale
150-	discount of 150
100I	new entry on PLU 100 (repetition is no more possible after discount)

The way of managing this translation may vary depending on the received string. Let analyse the action to be performed in the PC program in order to obtain the previously described transaction modification in the three possible events:

CASE OF EVENT 1

ACTION	STRINGS RECEIVED OR SENT BY THE PC	STRINGS ECHOED BY THE DRIVER	COMMENTS
read	2.	2	set quantity to 2 item entry not completed end of the string reached no need of reply
read	5*	.5	set quantity to 2.5 item entry not completed end of the string reached no need of reply
read	1000R	*1000	this item require no action end of the string reached no need of reply
read	100I	R100	set q.ty to 1 and add it on PLU 100 sale counter counter=1 implies no discount end of the string reached no need of reply
read	I	I	set q.ty to 1 and add it on PLU 100 sale counter counter = 2 implies no discount end of the string reached no need of reply
read	I	I	set q.ty to 1 and add it on PLU 100 sale counter counter = 3 implies discount builds the reply string I150- end of the string reached
write	I150-		reset PLU 100 counter to 0
read	I		set q.ty to 1 and add it on PLU 100 sale counter counter=1 implies no discount repetition after discount needs the PLU number to be declared builds the reply string 100I end of the string reached
write	100I		

CASE OF EVENT 2

ACTION	STRINGS RECEIVED OR SENT BY THE PC	STRINGS ECHOED BY THE DRIVER	COMMENTS
read	2.5*1000R100IIII	2	analyses the string
extracts	2.		set quantity to 2 item entry not completed string not finished copy sequence to reply string 2.
extracts	5*		set quantity to 2.5 item entry not completed string not finished add sequence to reply string 2.5*
extracts	1000R		this item require no action string not finished add sequence to reply string 2.5*1000R
extracts	100I		set q.ty to 1 and add it on PLU 100 sale counter counter=1 implies no discount string not finished add sequence to reply string 2.5*1000R100I
extracts	I		set q.ty to 1 and add it on PLU 100 sale counter counter = 2 implies no discount string not finished add sequence to reply string 2.5*1000R100II
extracts	I		set q.ty to 1 and add it on PLU 100 sale counter counter = 3 implies discount modify the sequence string not finished add sequence to reply string 2.5*1000R100III150-
extracts	I		set q.ty to 1 and add it on PLU 100 sale counter counter=1 implies no discount repetition after discount needs the PLU number to be declared modify the sequence add sequence to reply string 2.5*1000R100III150-100I end of string reached the string has been modified and must be sent cut the already echoed numbers
write	.5*1000R100III150-100I		

CASE OF EVENT 3

ACTION	STRINGS RECEIVED OR SENT BY THE PC	STRINGS ECHOED BY THE DRIVER	COMMENTS
read	2.5*10	2	analyses the string
extracts	2.		set quantity to 2 item entry not completed string not finished copy sequence to reply string 2.
extracts	5*		set quantity to 2.5 item entry not completed string not finished add sequence to reply string 2.5*
extracts	10		this item require no action string not finished item entry not completed end of the string reached no need of reply store numeric keys to analyse them with the next sequence
read	00R	.5*1000	analyses the string
analyses	1000R		this item require no action end of the string reached no need of reply
read	100IIII	R	analyses the string
extracts	100I	100	set q.ty to 1 and add it on PLU 100 sale counter counter=1 implies no discount string not finished copy sequence to reply string 100I
extracts	I		set q.ty to 1 and add it on PLU 100 sale counter counter = 2 implies no discount string not finished add sequence to reply string 100II
extracts	I		set q.ty to 1 and add it on PLU 100 sale counter counter = 3 implies discount modify the sequence string not finished add sequence to reply string 100III150-

ACTION	STRINGS RECEIVED OR SENT BY THE PC	STRINGS ECHOED BY THE DRIVER	COMMENTS
extracts	I		set q.ty to 1 and add it on PLU 100 sale counter counter=1 implies no discount repetition after discount needs the PLU number to be declared modify the sequence add sequence to reply string 100III150-100I end of string reached the string has been modified and must be sent cut the already echoed numbers
write	III150-100I		

Note that when linking the basic sequences to build the reply string, it must be taken into account that the numeric characters preceding the first non numeric have already been echoed.

Using the Bar Code

When inputting a sale using the bar code reader connected to the ECR, the sale sequence sent to the PC. The PC program has to take into account that a code not readable with the bar code reader is entered from the keyboard with the same sequence used for PLU entry, where the PLU number is replaced by the code. Following are examples of valid bar coded item entries:

BAR CODE ENTRY	MEANING
“12345678”	sale of a quantity of 1 on the article having code 12345678
2*“A7610424288347”	sale of a quantity of 2 on the article having code 7610424288347 (the letter A is a code type identifier usually added by the bar code reader)
-3.5*0051111128831I	void of a sale of a quantity of 3.5 on the article having code 0051111128831

If the article code is present into the ECR memory, the PC has to convert the Bar Code, received as a string, into a PLU sale sequence, using only the numerical part of the code, since the code stored into the PLU’s is only numeric.

In example, when receiving the code **“A7610424288347”**, the PC must send to the ECR the sequence **7610424288347I**.

If the code is received as a PLU entry sequence, there is no need for the PC to modify the sequence itself. If the article code is not present into the ECR memory, to perform the sale the PC must drive the ECR modifying the bar code sale sequence with a similar sequence that performs the sale on a department or on a PLU.

Obviously, beside sales sequences, also refund and void sequences using the bar code reader must be expected and managed by the PC.

The ECR protocol provides also the host computer with the capability to replace the item description printed on the receipt with a new one, valid only for the item and its repetition, allowing the appropriate article description to be printed.

To make the PC item description (of maximum 11 characters) to be printed, this must be sent, enclosed within double quote, just before the whole item entry sequence. As consequence of this, to print the proper item description it is needed to clear on the ECR side basic sequences of the bar code item entry already echoed. For this purpose, in the BARCODE.PAS example program, each reply to a bar coded entry starts with a clear key (K).

To update the stock quantity in the PC articles data base, it is necessary to keep track of the quantity sold for each article. To do this, the sales quantity (1 if not declared or on item repeat) must be extracted from the keyboard sequences, and added (when sale or correction of a refund) or subtracted (in case of refund or correction) to a sold quantity counter.

The following examples are based on the BARCODE.PAS mode of operation. This program loads from a file into a memory buffer a very simple articles database, containing for each article the following data:

Contents	Format
Numeric code	(up to 14 digits)
description	(up to 11 characters)
price	(up to 9 digits)
linked department or PLU	(*)
sold quantity counter	

(*) when building the reply message, numbers from 1 to 8 are treated as departments, numbers from 9 to 2000 are treated as open PLU.

When receiving sequences from an ECR, the program analyses them looking for significant key codes:

Key code	Meaning
-	if not preceded by digits, or by a % key, it identifies a negative item entry (refund or void).
.	the preceding digits are the integer part of a percentage or of a sale quantity
*	the preceding digits, eventually queued as decimal part to the integer part entered with the . key, are the sale quantity.
I	the preceding digits are the code of a PLU (if between 1 and 2000) or an article code (if greater than 2000. If not preceded by digits, it identifies the repetition of the last PLU or bar code sale with unit quantity. To be acceptable, the repeat sequence must be entered right after an integer quantity sale or another repeat.
L	if not preceded by digits, it identifies the correction of the previous item entry. The correction is valid no other keys are entered between the item and its correction. If preceded by digits, they are the price to be entered on an open PLU.
K	it clears a partially entered item entry.
%	it identifies a percentage. In this context it tested to discriminate between discounts and refunds.

While analysing the received messages, depending on the key sequence the program sets some status flag in order to process the items in the proper way.

To make some example, let suppose that exist the following articles:

CODE	DESCRIPTION	PRICE	LINKED TO
7610424288347	COCACOLA 1L	3500	department 5
24570004	APPLES	1200	open PLU 130

Under this hypothesis, taking into account everything said before, the received sequences must be translated in the following way (empty reply message indicate no need to reply):

RECEIVED MESSAGE	REPLY MESSAGE	ACTION TAKEN / NOTES
2*“7610424288347”	K"COCACOLA 1L"2*3500T	counter + 2
I	T	counter + 1
I	T	counter + 1
L		counter - 1
1500R3.5*“24570004”	RK"APPLES"3.5*1200L130I	counter + 3.5
I		(repeat not allowed after decimal quantity)
7610424288347I	K"COCACOLA 1L"3500T	counter + 1
K		
L		(correction not allowed after clear)
-2*7610424288347I	K"COCACOLA 1L"-2*3500T	counter - 2
L		counter + 2

Sending Item Descriptions

In the bar code management section is described how it is possible to send an Item Description to the ECR.

To properly use this feature, it is necessary to follow some rules.

It is very important never to exceed 11 characters of description length. If such event occurs, the ECR will reject the item entry, sending the communication error character (G) and then starting a new communication session.

If the description contains special characters (other then the ASCII numbers 0 to 9 or capital letters A to Z) the description string must be translated in such way that the sent characters are according to the ECR character set.

The description string modification is obviously function of the character set installed in the ECR to specialise it to a country.

Starting from the ECR Character Codes reported in Appendix A of the User Manual, that contains the numerical code of the ECR characters, it is possible to translate a given character of the description into the character to be sent in the following way: the character to be sent is the PC character having as numerical code the ECR character code corresponding to the given character + 31. For instance, the ECR code for the character 0 is 17 and the character to be sent must have code 17+31=48 (ASCII code for " 0"); the ECR code for the character É is 69 and the character to be sent must have code 69+31=100 (ASCII code for "d").

The numerical code corresponding to a given PC character is the numerical representation of its coding (usually ASCII for the standard alphabet), that is the number to be inserted in the numeric key-pad while pressing the ALT key to enter the character (refer to the DOS instruction manual). For in-stance, for standard PC the numerical code for the character 0 is 48 (it may be entered pressing ALT 48); for the character É is 144 (it may be entered pressing ALT 144); the code 100 corresponds to the character d.

Following the previous example, the description "0É" must be translated as "0d".

The simplest way to translate the descriptions in a program is to get the numerical code of a character and use it to access a look-up table that contains the translated character.

Following is an example of a Pascal function that perform the translation of a string and its limitation at 11 character maximum:

```
{ TRANSLATION TABLE FROM PC CHAR CODE TO ECR CHAR CODE }
const ToEcrChar : array [0..255] of char =
    (' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ','!','"',' ',' ','%','&','''','(',')','*','+',',','-','.','/',
     '0','1','2','3','4','5','6','7','8','9',':',' ','<','=','>','?',
     ' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
     'P','Q','R','S','T','U','V','W','X','Y','Z',' ',' ',' ',' ',' ',
     ' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
     'P','Q','R','S','T','U','V','W','X','Y','Z',' ',' ',' ',' ',' ',
     'b','z','d',']','`','\','_','b','f','g','e','k','j','i','`','_',
     'd',' ',' ','q','s','p','y','x','|','s','z',' ','m',' ',' ',' ',
     '[','h','o','w',' ',' ',' ',' ','$',' ',' ',' ',' ','#',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ','~',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',';',' ',' ',' ',' ',' ',' ',' ');


function Translate (InputDescr : string) : string;

var i,j  : integer;
    code : integer;
    OutDescr : string;

begin
  OutDescr:='';
  j:=length(InputDescr);           { GET THE INPUT STRING LENGTH }
  if j > 11 then j:=11;            { MAX LENGTH ALLOWED IS 11 }
  for i:=1 to j do                 { FOR EACH CHARACTER }
      begin
         code:=ord(InputDescr[i]);              { GET ITS NUMERICAL CODE}
         OutDescr:=OutDescr + ToEcrChar[code];  { ADD ITS TRANSLATION }
                                                { TO THE OUTPUT STRING }
      end;
  Translate := OutDescr;           { RETURNS THE TRANSLATED STRING }
end;
```

The array ToEcrChar must change according to the ECR programmed language and character set.

Printer Messages

If the option to send to the PC echo of all printer operations(printouts and line feeds) has been set on the ECR, each time that a printer operation is performed a printer message is sent.

Each printer message has the following structure:

```
<Command> <Data>

where <Command> is a single character indicating the operation
                performed on the printer (Normal print, Double
                heigth print, Line feed);
      <Data>    present only after print type commands, it
                contains the 18 or 24 character (depending on the
                ECR model) printed on the receipt.

The possible command codes are: a, b, c, d, e, f, g, h.
The used are:
    b = normal print
    d = double height
    h = line feed
```

Even when printer messages are received, each read request to the driver will return only one printer message. This is valid also when printer messages are received mixed with keyboard messages.

An example of possible message reception sequence is:

```
3*
1000R
b    3 x 1.000
bDEPARTMENT3 C      3.000
R20
bDEPARTMENT3 C      1.000
00P
bDEPARTMENT1   A    2.000
X
h
dTOTAL              6.000
```

What explained before for the item description strings, is applicable to the Data field of the printer messages: the characters sent by the ECR are corresponding to a numerical code equal to the ECR internal code + 31.

On the PC, the printed strings should be translated to the correct PC character coding. This can be done in the same way as previously described, but using a different translation look-up table, always function of the ECR language and character set.

For example, the reverse translation table corresponding to the previous example is:

```
{ TRANSLATION TABLE FROM ECR CHAR CODE TO PC CHAR CODE }

const ToPcChar : array [0..255] of char =
    (' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ','!','"','í','è','%','&','''','(',')','*','+',',','-','.','/',
     '0','1','2','3','4','5','6','7','8','9',':','°','<','=','>','?',
     ' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
     'P','Q','R','S','T','U','V','W','X','Y','Z','à','Å','Ã',' ','Æ',
     'Ä',' ','À',' ','Â','Ê','È','É','á','Í','Ì','Ë',' ','Ü',' ','â',
     'Õ','',' ','Ô',' ',' ',' ','ã','×','Ö','Á',' ','Ø',' ','ß',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ');
```

Error Messages

The error messages are sent by the ECR whenever an error condition has been detected while executing a keyboard command (both entered through keyboard or sent by the PC).

An error message is a 4 character strings. First and last characters are the letter E. The mid two character are the numerical error code, that is shown also on the display.

Example of error message is:

E10E

When controlling the ECR from the PC, error messages can be monitored to verify that a keyboard message has been accepted by the ECR.

READING ECR MEMORY

It is possible, through some function, to ask the ECR to send part of its memory content (consult the User Manual for more details on function key sequences). Following is a list of the main functions to read the ECR memory data.

Function	Data transferred	Command Sequence
920	Departments	920=15290=
921	Cashiers	921=15290=
922	Header + Trailer	922=15290=
923	Alternative Trailers	923=15290=
924	ECR Internal Data	924=15290=
925	ECR Totals Memory Area	925=15290=
926	ECR Configuration Data	926=15290=
927	PLU 1 to 512	927=15290=
928	PLU 513 to 1024	928=15290=
929	PLU 1025 to 1536	929=15290=
930	PLU 1537 to 2000	930=15290=
940	Single PLU	940=15290=<plu number>=

The memory content is transmitted as a single memory message, having the structure:

```
<ID Char> <Mem Data>

where <ID Char>  is the character &;
      <Mem Data> is a sequence of bytes.
```

The length of the <Mem Data> field is equal to the dimension of the requested memory area. For example, each PLU data area is 16384 bytes long.

Two events are possible when performing the reading request to the driver:

- 1 - the dimension of the driver interface structure is big enough to contain all the memory message, in which case the read returned code will be 0;
- 2 - the dimension of the driver interface structure is not enough to contain all the memory message, in which case the driver will fill up the interface structure with part of the memory message, and the read returned code will be 9.
- If the second event occurs, it is possible to read again the driver to get the following part of the memory. When the last part of the memory message is read, the returned code will be 0. Also partial memory content will start with the <ID Char>.

Once the memory content has been received, to extract the desired data the program must interpret the memory image decoding it according to the ECR internal data structure.

Following is reported the structure of the memory area of main interest.

Data Read from the ECR Memory

DEPARTMENTS

The memory data will contain 5 or 8 or 16 departments records, according to the ECR model, each one containing:

Description	11 characters (to be translated as previously explained)
Option	byte : bit 0,1,2 = VAT pointer (0 to 7) bit 3 = unused bit 3 = Single Item bit 5..7 = unused
Preset Price	4 bytes integer
Sales Total	4 bytes integer
Sales Counter	2 bytes integer
Spare	9 bytes
Entry Limit (Spare if not present)	1 byte = maximum figures number accepted as unit price from keyboard

CASHIERS

The memory data will contain 20 cashier records, each one containing:

Name	Up to 20 characters depending on the ECR model (to be translated as previously explained)
Gross Sales	6 bytes integer (less significant byte is received first)
Loan Total	6 bytes integer
Pick-up Total	6 bytes integer
Cash Total	6 bytes integer
Check Total	6 bytes integer
Tend. 3 Total	6 bytes integer

HEADER + TRAILER

The memory data will contain the characters of the header lines followed by the trailer line (to be translated as previously explained). The number of header lines and number of characters per lines depend on the ECR model. For ECR with 15 characters thermal printer, the content will be:

Header Line 1	15 characters
Header Line 2	15 characters
Header Line 3	15 characters
Header Line 4	15 characters
Header Line 5	15 characters
Header Line 6	15 characters
Header Line 7	15 characters
Trailer line	15 characters

ALTERNATIVE TRAILER

The memory data will contain the characters of the ten alternative trailer lines (to be translated as previously explained). The number of characters per lines depends on the ECR model. For ECR with 15 characters thermal printer, the content will be:

Alternative Trailer 1	15 characters
Alternative Trailer 2	15 characters
.	.
.	.
Alternative Trailer 9	15 characters
Alternative Trailer 10	15 characters

ECR Internal Data

This memory area contains internal parameter data, not all of them are of interest to an external user. The interesting data of this area are the following:

Start Byte Position	Meaning	Format
1	First Data Byte of Dump 924	
84	Option Bits	8 bits (bit 0 is the lsb) bit 0 = Discount Disabled bit 1 = Mark-up Disabled bit 2 = Time on display Disabled bit 3-7 = N.A.
85	Preset Discount %	1 byte integer
86	Preset Mark-up %	1 byte integer
141	VAT Rates	8 integers values (2 bytes each) representing the VAT rates multiplied by 100
290	Enabled VAT Rates Mask	8 bits (lsb enables VAT 1, etc.)
644	Status Bits	8 bits (bit 0 is the lsb) bit 0-6 = N.A. bit 7 = Fiscalized ECR
646	Number of Active Cashiers	1 byte integer
675	General Options	1 word (its meaning is explained in the programming manual)

ECR Totals Memory Area

The memory data will contain several internal ECR data (including workspace areas). The interesting part of these data are the Daily Totals area. It starts in correspondence of the 79th received byte of the Mem Data Field and is structured as follows:

Discount Total	6 bytes integer (less significant byte is received first)
Mark-up Total	6 bytes integer
Void Total	6 bytes integer
Refund Total	6 bytes integer
Discount Counter	2 bytes integer
Mark-up Counter	2 bytes integer
Void Counter	2 bytes integer
Refund Counter	2 bytes integer
Loan Total	6 bytes integer
Pick-up Total	6 bytes integer
Cash Total	6 bytes integer
Check Total	6 bytes integer
Tender 3 Total	6 bytes integer
Cash Counter	2 bytes integer
Check Counter	2 bytes integer
Tender 3 Counter	2 bytes integer

ECR Configuration Data

This memory area contains internal parameter data, not all of them are of interest to an external user. The interesting data of this area are the following:

Start Byte Position	Meaning	Format
1	First Data Byte of Dump 926	
3	Number of VAT Rates	1 byte integer (0 to 7 for 1 to 8 VAT Rates)
16	Status Bits	8 bits (bit 0 is the lsb) bit 0-3 = N.A. bit 4 = VAT Exempt present bit 5-7 = N.A.
21	Configuration Bits	8 bits (bit 0 is the lsb) bit 0 = N.A. bit 1 = PLU records with EAN Code bit 2 = N.A. bit 3 = Departments with entry limit bit 4-7 = N.A.
25	Fixed General Options Mask	1 word (it contains the bit mask of the options that cannot be modified on fiscalized ECR)
29	Number of departments	1 byte integer
30	Number of PLU	2 bytes integer
32	Characters per printer line	1 byte integer
33	Number of Header Lines	1 byte integer
34	Characters per Cashier Name	1 byte integer

PLU

The memory data will contain up to 512 PLU records, each one containing:

Description	11 characters (to be translated as previously explained)
Option	byte : bit 0,1,2,3 = Major Department pointer, for a maximum of 16 departments (0 to 15), depending on ECR model. bit 4 = Single Item bit 5 = Open Price bit 6..7 =0 unused NOTE: For Poland ECR, the first four bits have the following meaning: bit 0,1,2 = VAT pointer (0 to 7) bit 3 = 0 unused
Preset Price (only if Preset Plu) Sales Total (only if Open Plu)	4 bytes integer
Sales Quantity	4 bytes integer (3 decimals fixed point)
Inventory	4 bytes integer (3 decimals fixed point)
EAN Code (Spare if not present)	7 bytes (BCD packed numeric code)
Spare	1 byte

SINGLE PLU

The memory data will contain a single PLU record, according to the previous described format.

- ***Notes : characters are coded as ECR numerical code + 31, to translate them into PC characters, the same conversion algorithm as for the print messages shall be used. - memory address range may change depending on the ECR model and revision.***

The program GETMEM.PAS is an example of how to read and decode ECR memory data.

ECR PROGRAMMING

For obvious security reasons, ECR memory data can be only read from the PC.

It is anyway possible to perform ECR data customisation (like description, price, option programming) sending from the PC the sequences reported into the Programming Manual.

Using the ECR as a PC Terminal

For some application, or during ECR programming, may be necessary for the ECR to be slave of the PC.

Depending on the Command Field within the structure that interfaces the driver, it is possible to request the driver for different mode of operation while requesting a read function.

The bit-4 of the Command field specifies whether the driver must return to the calling program only if a key sequence has been received from the ECR (bit-4=0) or also when the ECR polls the PC to ask for a remote key sequence (bit-4=1).

Before sending from the PC a key sequence, which is not a reply to are received keyboard sequence, it is necessary to wait for the ECR polling, in order to synchronise the transmission with the internal status of the ECR.

To do this the program must request the read function to the driver with the bit-4 of the Command field set, wait for the reception of an empty sequence (Length field = 0) and then send the key sequence to the ECR.

The program EXAMPLE2.PAS shows how to do this operation.

In case that it is desired an exclusive PC control over the ECR, that means a complete disabling of the ECR keyboard, it is possible to ask the driver not to send echo of numeric keys and then, each time a not empty sequence is received from the ECR, reply with an empty sequence.

To ask the driver not to send echo of numeric keys, the bit-7 of the Command field during the read operation.

Following is an example of such a procedure, that locks the ECR until a key is pressed on the PC keyboard:

```
procedure EcrKeyLock;

{ Rx and Tx buffers definition }

type  ComBuff = record
                  Command   : integer;
                  Length    : integer;
                  Data      : array [1 .. 200] of char;
                end;

var  TxRxBuff : ComBuff;

{ constant code definition }

const  CodeOK       =    0 ;
       PcKeyPressed =    8 ;
       NoNumberEcho =  $80 ;
       PcKeyCtrl    =  $40 ;

begin
    repeat
        TxRxBuff.Command:=PcKeyCtrl + NoNumberEcho;
        TxRxBuff.Length:=200;
        read (Ecr,TxRxBuff.Command);
        if (TxRxBuff.Command=CodeOK)
           then begin
             TxRxBuff.Length:=0;
             write(Ecr, TxRxBuff.Command);
           end;
    until TxRxBuff.Command=PcKeyPressed;
end;
```