BasicCard POS

Basic Programmable POS Terminal for BasicCard

Document version 1.03
22. October 2002

Author: Michael Petig
email: development@zeitcontrol.de

Siedlerweg 39
32429 Minden
Germany

Tel: +49 (0)571-50522-0
Fax: +49 (0)571-50522-99
Web sites:
http://www.zeitcontrol.de
http://www.basiccard.com

Contents

1. Overview

The BasicCard POS is a **P**oint **O**f **S**ale terminal which can be programmed by use of Basic programming language. It is designed to be used with ZeitControl BasicCard (see http://www.basiccard.com). BasicCard is a Basic programmable real smart card with support for encryption algorithms like DES or AES. For developing applications for the BasicCard POS you need the same set of tools as for BasicCard itself. You can download these tools for free in “Free Download” area at http://www.basiccard.com. Here you must download and install the “BasicCard Development Software ...”. The development software contains a Basic compiler which can generate code for the BasicCard itself or for a terminal using BasicCards. The compiled terminal program runs on a Windows PC, it uses either a ZeitControl card reader or any PC/SC compliant card reader to access a BasicCard. The same terminal programs can be converted and loaded into the BasicCard POS.

2. Content of the BasicCard POS Development Kit

The development kit contains the following:

- 1 CD containing BasicCard POS Development Software and latest version of BasicCard Development Software
- 2 BasicCards ZC3.9 with Debit example loaded
- 1 BasicCard POS device (with Debit example loaded)
- 1 PIN Pad device
- 1 Power supply
- 1 Serial cable
- 1 Hardware manual

The development kit does not contain the normal BasicCard development tools (e.g. smart card reader) which are required. Such it is recommended to order a BasicCard Development Kit as well if this kit is not already purchased.

3. Software Installation

First install standard BasicCard Development Software. When finished install BasicCard POS Development Software.

4. Limitations

Currently only T=1 is supported. If you want to use the BasicCard POS with Professional BasicCards you must program the cards to use T=1 instead of default T=0 protocol.

File system functions for using files on POS or BasicCard are not supported by current version of BasicCard POS. File system function can only be used to access POS devices such as printer or PIN Pad.

5. Developer Guide

Please refer to BasicCard manual for guidance in how to develop BasicCard and using terminal applications. Then follow additional guidelines here. A printed version of BasicCard manual is part of standard BasicCard Development Kit. The BasicCard Development Software included with this BasicCard POS Development Kit contains the BasicCard manual as PDF file available through “Start”, “Programs”, “BasicCard Pro”, “BasicCard Manual”.

5.1 Tools

Additionally to standard BasicCard tools some extra tools are required. These tools are part of BasicCard POS Development Software. Once installed you find these tools in directory c:\basiccardpro\pos\.

5.1.1 IMGCONV.EXE

Converts a BasicCard terminal image (.img or .dbg file) into a binary format file (BcImg.bin) loadable into BasicCard POS. In the example directories for BasicCard POS you find a batch file CONV.BAT which can be used for running IMGCONV.EXE.

Syntax: **IMGCONV** [*param* [*param* ...]] *image-file* [*param* [*param* ...]]

image-file	The image file output by the compiler. If no extension is supplied, *image-file***.img** is assumed. (A debug file is also allowed here; in this case the **.dbg** extension must be suplied.)	
param	One of the following:	
	-D	Displays the conversion status while executing
	-O*outfile*	Change the name of the output file. By default the result is stored in file **BcImg.bin**. If no extension is supplied **.bin** is assumed.

5.1.2 TXFILE.EXE

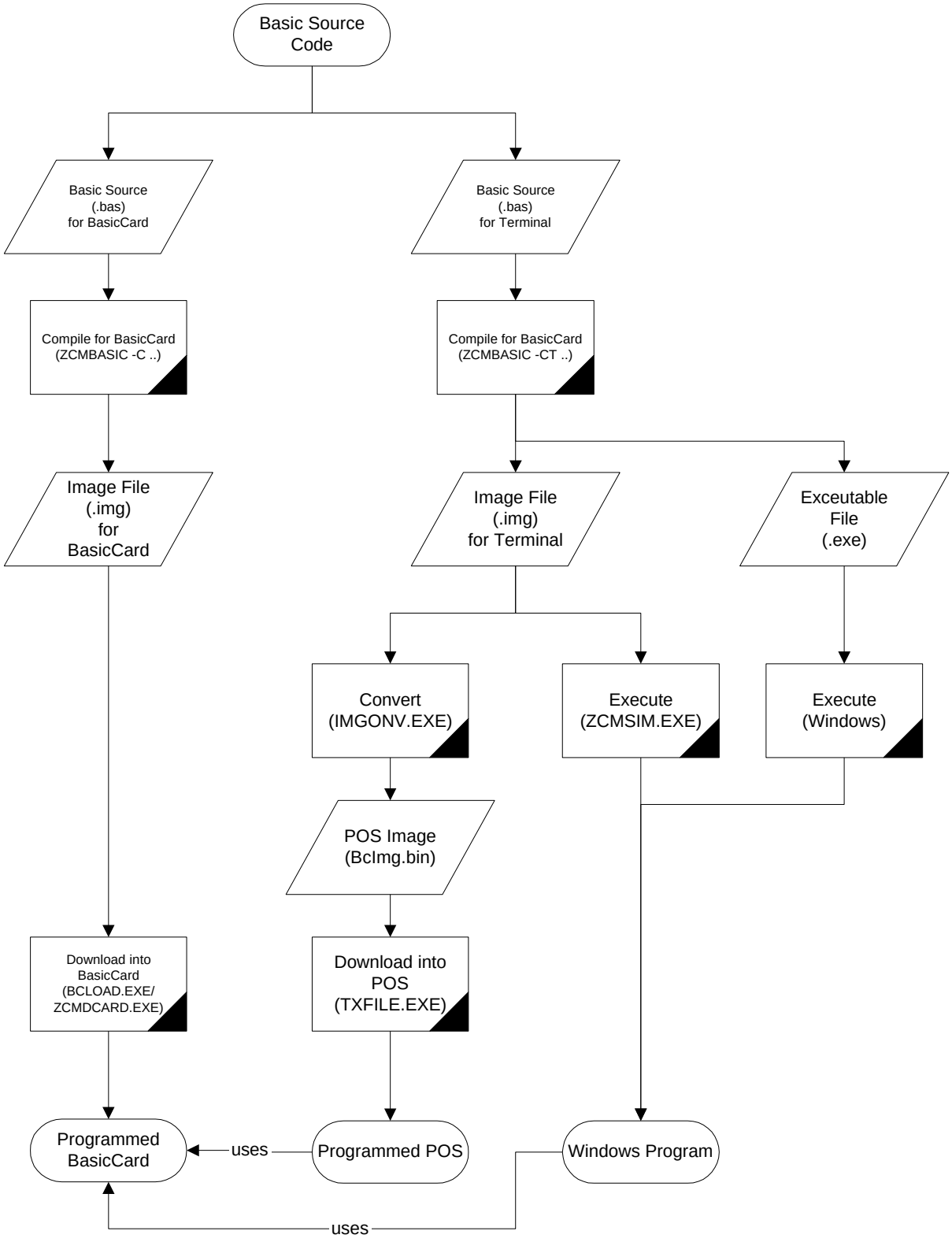
Tool to load converted image (BcImg.bin) into BasicCard POS. For this you must connect the serial port of the BasicCard POS to a serial port of your computer. You will find a batch file DLIMG.BAT as part of tools for BasicCard POS. You may use and modify this for your purpose. Please note: You must call DLIMG.BAT followed by target port, e.g. **DLIMG COM1**.

Syntax in general: **TXFILE** *filename COM-Port BaudRate Offset Type Title*

filename	The file load into BasicCard POS. For Basic files this must be a converted image file created by **imgconv.exe**.
COM-Port	The comport the BasicCard POS is connected to. E.g. **COM1**.
Baudrate	The baud rate used for transmitting file this must be **19200**
Offset	The offset at which the file should be loaded into BasicCard POS memory. This must be **80000** for Basic programs.
Type	The file type to be loaded. This must be **3** for Basic programs.
Title	The title which is stored to name this file in BasicCard POS as text string in “quotes”, e.g. **“Basic IMG”**.

5.2 Workflow for Creating BasicCard Solutions, Workflow

Following scheme should explain to you the process of creating a BasicCard solution. Leftmost column shows the process for creating your BasicCard program itself, which finally results in a programmed card. Columns on right side show process of generating a terminal program which finally uses the programmed card. You may either generate an Windows executable (.exe) which can be executed directly from Windows or a terminal image file (.img) which can be executed by use of ZCMSIM.EXE. So this both finally results in a Windows Program. To create a POS program the terminal image file must be converted the resulting file (BcImg.bin) then can be loaded into the POS which will result in a programmed POS.



5.3 Usage

5.3.1 Creating a BasicCard POS Program

You create a BasicCard POS program same way as if you create a normal Windows PC terminal program using our BasicCard development tools. This means you can test and debug your program on your computer first. You should do this since there are no debugging capabilities available for BasicCard POS itself. See BasicCard manual for how to develop your BasicCard and terminal application.

There are some differences in between running a terminal program on Windows PC and BasicCard POS. See references at end of this document for this.

5.3.2 Downloading a Basic Program

Once you have created your Basic terminal program you should first test it using terminal debugger on your Windows PC. If everything runs as expected, you can download it into BasicCard POS. For this follow these guideline.

- Switch off the BasicCard POS. If you cannot switch off by function inside previously loaded program, use hidden reset switch on backside of device (left to product label).
- Connect the BasicCard POS to a free serial port of your computer with connection cable included.
- Hold key 1 pressed while switching on BasicCard POS by use of red power on button.
- A menu appears, here choose F2 “Download App”.
- Confirm next screen with enter.
- Now download file as described in description of TXFILE utility.
- Once the file is completely loaded press enter.
- Then follow guidelines as shown in BasicCard POS display.

5.3.3 Running your BasicCard Program

Once your Basic program is loaded it is automatically executed if the device is started or if you choose “Basic VM” from “Run App”.

5.4 Step by Step Example with Calculator Example

5.4.1 Start with Calculator Example

- Open ZeitControl Professional IDE by “Start”, “Programs”, “BasicCard Pro”, “ZeitControl Professional IDE”.
- Now select menu “Project”, “Open” and browse to c:\basiccardpro\pos\examples\calc\calc.zcp. Click “Open” to open the project.
- Now you may compile the project file (see “Project”, “Compile..” menu) and start the debuggers (see “Project”, “Start...” menu). You must start the card debugger to download the card program into a BasicCard.
- To view the source codes click menu “File”, “Open” and open the .bas files (see file list below for description of files).
- Click “Project”, “Compile All” to compile all sources. You must confirm all popping up dialogs to continue to the end. These dialogs will show to you either compiler errors, or resulting size of compiled sources.

5.4.2 Simulate and Debug the Example

- Follow all steps of “Start with Calculator example” as described before.
- Start the card debugger by click “Project”, “Start BasicCard”, “Calc.ZCC”. The card debugger will appear.
- Return to IDE and start the terminal debugger by click “Project”, “Start Terminal”, “calctest.ZCT”.
- Rearrange all windows (card debugger, terminal debugger) so you can view all windows.
- Select the terminal debugger. Note: By default the card debugger is operating as virtual card reader at virtual comport 201 and the terminal debugger is using this virtual card reader.
- Use buttons “Step Over” and “Step Into” to step through program code line by line. If you use “Step Into” you will step into Basic subs, functions and commands, while “Step Over” will always step to next execution line without stepping into subs, functions or commands. Use Button “Run” to make the program execute continuously. If you step into a command the control will move from terminal debugger to card debugger.
- You may set breakpoints by clicking on blue pin heads on left side of source code inside each debugger. Breakpoints are then marked with red pin heads. Clicking on a breakpoint mark will remove the break point.
- You may evaluate and view variables, by marking the variable name by use of mouse. Once you finish marking a word (release left mouse button), the “Evaluate Expression” window will appear. You may just close the window, or add the variable to your watch list.
- To get more used to card and terminal debugger you should play with all this functions and other features available through menus of debuggers.

5.4.3 Load the Card Program into a Real BasicCard

- Follow all steps of “Start with Calculator example” as described before.
- Start the card debugger by click “Project”, “Start BasicCard”, “Calc.ZCC”. The card debugger will appear.
- Click menu “File”, “Compile”. Check the “Card type” to fit your BasicCard available. If card type does not fit, change card type and then press compile. Close dialog box with compile result.
- Close compile (“BasicCard Program Options”) dialog by press OK.
- Click menu “Card”, “Download to Real Card...”.
- Choose the “COM Port” to select your smart card reader. If you do not now which is the proper setting for “COM Port” try different settings until you have success with next steps.
- Click “Download” button.
- When requested insert your BasicCard.
- If everything is completed you now have your first programmed BasicCard. Press “Done” to close the dialog.
- Close the card debugger.

5.4.4 Debug the Terminal with real BasicCard

- Create a real BasicCard with Calculator example by follow all steps of “Load the Card Program into a Real BasicCard”.
- Change to “ZeitControl Professional IDE”.
- Start terminal debugger by use of menu “Project”, “Start Terminal”, “calctest.ZCT”
- By default the example uses the simulated card in virtual card reader (COM port 201). To change this click menu “Options”, “COM Port...”. Choose the “COM port” to fit your physical card reader (same value as you have used to load the BasicCard).
- Press OK to close the dialog.
- Continue same as in “Simulate and Debug the Example”. In opposite to debug operation with simulated card you cannot step into the card source code.

5.4.5 Convert and Download the Terminal into BasicCard POS

- Prepare the BasicCard POS for download by
  - Switch off the BasicCard POS. If you cannot switch of by function inside previously loaded program, use hidden reset switch on backside of device (left to product label).
  - Connect the BasicCard POS to a free serial port of your computer with connection cable included.
  - Hold key **1** pressed while switching on BasicCard POS by use of red power on button.
  - A menu appears, here choose F2 “Download App”.
  - Confirm next screen with enter.
- Follow all steps of “Start with Calculator Example”.
- Open a command line window. This depends on your Windows version. E.g. for Windows XP, 2000, NT use “Start”, “Run” then enter “cmd” and press OK. E.g. for Windows 95, 98, ME use “Start”, “Run” then enter “command” and press OK.
- Change to example directory in command line window by enter (confirm each line with Enter)
  - c:
  - cd c:\basiccardpro\pos\examples\calc
- Convert previously compiled terminal image for use with BasicCard POS by enter
  - conv
- Start download to BasicCard POS by enter “dlimg comX”, where X is the serial port number the POS is connected to. If you do not know which COM port is used try all values from COM1 up to COM4. E.g.:
  - dlimg com2
- Once the file is completely loaded press the enter button of POS device.
- Then follow guidelines as shown in BasicCard POS display.
- Now your POS is loaded with calculator example. Choose “Run App” and “Basic VM” or just switch device on again to run the application. Use it with your BasicCard you have created before.

6. Examples

In principle all examples of BasicCard Development Software can be used with BasicCard POS as well. But since the BasicCard POS has some built in units which are not available on Windows PC and since the display and keyboard of the POS is limited compared to the PC, there is a small set of examples include with POS Development Kit. These special examples are prepared to use the POS in an optimal way.

6.1 Calculator

This example is identical to same named example of BasicCard development software, but there are some small modifications compared to original one. The Calculator example does some calculation inside BasicCard. So first some automatic test calculations are done, then you may enter your own input to be calculated. Note: The Calculator example is for learning, such it is not a typical smart card application as needed in real life.

Following files are included in c:\basiccardpro\pos\examples\calc:

File Name	Description
calc.zcp	Project file for use by ZeitControl Professional IDE.
calctest.zct	Terminal file (project file containing information about terminal related files).
calc.zcc	Card file (project file containing information about card related files).
calc.bas	Basic source code containing card application (to run inside BasicCard).
calc.def	Basic source code containing command declarations and other global source code.
calctest.bas	Basic source code containing terminal application (to run on PC or inside BasicCard POS).
calckeys.bas	Basic source code containing encryption keys for terminal and card.
compile.bat	Batch file to run compiler and compile all source files. You may use this on command line, but it is recommended to use the compile function of ZeitControl Professional IDE or card or terminal debugger instead.
conv.bat	Batch file calling converter to convert compiled image into binary image for BasicCard POS.
dlimg.bat	Batch file to download binary image into BasicCard POS

6.2 Debit

The debit example shows how to use a BasicCard for a payment card. Inside the BasicCard the value and an ID is stored. Access to the card is secured by DES encryption. Further many operations require a PIN number, which must be entered by the card owner. There are three programs required for this application. The card program itself, which contains the value and secures operations on this value. The issuer program to initialize the card with an initial value, the user ID and the PIN number. And finally the debit program to make pay and load operations by removing or adding a value from/to the card. The issuer program requires character inputs e.g. for entering the name of the card owner. Because of this and the fact that these program needs to be run one time only for each issued card, this program is not customized for the BasicCard POS. So it should not be downloaded into the POS, instead it should run on the computer only. The dealer program, compared to same named example of BasicCard development software, is modified a lot. Many modifications was made to dealer.bas to make this program take the maximum use of all features of the BasicCard POS. This causes the source file to be much more complicated than for example the issuer.bas source file. But as an advantage, you may use this example with very few modifications as your final application. Most important change you have to do is to change the DES keys, to make your payment card secure. Please also keep in mind only cards in RUN state are secure, so all issued cards should be in RUN state. Still you must be careful with RUN state once a card is in RUN state there is no way back, and you cannot download another or modified program anymore.

Even if a lot of things are changed, the POS version of debit example is still compliant to other versions of this example in development software and API. Such you may take a look into BasicCard API examples as well and e.g. use the issuer program from Visual Basic example which give to you a more convenient user interface. Of cause if you have changed the keys you must change the keys of the examples as well.

Following files are included in c:\basiccardpro\pos\examples\debit:

File Name	Description
debit.zcp	Project file for use by ZeitControl Professional IDE.
issuer.zct	Issuer terminal file (project file containing information about issuer terminal related files).
dealer.zct	Dealer terminal file (project file containing information about dealer terminal related files).
debitcrd.zcc	Card file (project file containing information about card related files).
debitcrd.bas	Basic source code containing card application (to run inside BasicCard).
debitcrd.def	Basic source code containing command declarations and other global source code.
issuer.bas	Basic source code containing the issuer terminal application (to run on PC or inside BasicCard POS).
dealer.bas	Basic source code containing the dealer terminal application (to run on PC or inside BasicCard POS).
issuer.key	Basic source code containing encryption keys for use by issuer terminal and card.
dealer.key	Basic source code containing encryption keys for use by dealer terminal and card.
compile.bat	Batch file to run compiler and compile all source files. You may use this on command line, but it is recommended to use the compile function of ZeitControl Professional IDE or card or terminal debugger instead.
conv.bat	Batch file calling converter to convert compiled image into binary image for BasicCard POS.
dlimg.bat	Batch file to download binary image into BasicCard POS

Before using this example in a real world application, you should check if the security available when using this application is enough to fit your security requirements. Here some things you should know and keep in mind. Crucial part of the security in this application are the encryption keys. These keys must be kept secret. But on the other side the keys must be know to the card (all keys), to the issuer program (all keys) and to the dealer program (dealer key only). It is not difficulty to archive security for keys stored in the card itself, as long as all cards containing this keys and are given to none trusted persons (e.g. the users) are in RUN state. In opposite full security could not be archived for the terminal programs. It is possible to extract the key from PC terminal programs or the program loaded into the POS. It is possible to make this attack more difficult by storing the key in a covered way, but it is not possible to archive full security for the key here. Because of this two different keys are used, one for issuer application, another one for dealer application. To keep this security enhancement each terminal program should contain only the keys it needs. Further only persons needs to use a certain terminal program and are trustworthy should get access to a program. Still the security may not meet your requirements. In this case you should use a solution where all keys are stored in smart cards (BasicCards) only and no key is stored in terminal program code. This can be archived by using a two card system where different cards for user and dealer manages transactions by exchanging encrypted or signed certificates.

6.2.1 Using the Debit Example.

When switching on the POS your are requested to insert a debit card. Instead of this you may press one of the following buttons:

- F1: A help screen with a list of functions is displayed.
- F2: A log with all stored transactions is printed.
- F3: The transaction log is cleared.
- F4: The device is powered off.
- Menu: A service menu with more functions is activated.

By pressing “Menu” you get access to following functions:

- F1: Set date and time of real time clock
- F2: Show date and time of real time clock.
- Cancel: Close this menu.

When you are in “Insert card..” screen and insert a debit card, following will happen:

- The card will be validated and the current value (Balance) as well as the user ID are read from the card.
- If the card is valid this information are displayed on POS display. If the PIN pad is connected these information are also displayed on PIN pad display.
- On POS you now can choose between PAY or LOAD operation by pressing either F1 or F2.
- Once you have pressed either F1 or F2, you are requested to enter an amount for this operation. Please note: The decimal separator (‘.’) can be entered by pressing the button “↑/”. The “Clear” button works same as the backspace button on your computer and such can be used to correct wrong inputs. To finally confirm your input you must press “Enter”. Instead of pressing “Enter” you may press “Cancel” to cancel the operation.
- If you have confirmed your input by pressing enter and the PIN pad is connected, the operation (Pay or Load) and the amount is shown on PIN pad. Further the customer is requested to enter the PIN into the PIN pad. The customer may enter the PIN and then “Enter” to confirm the operation or he may press “Cancel” if he disagrees and wants to cancel the operation. While entering the PIN he may use the “Clear” button to correct wrong inputs. If no PIN pad is connected the same procedure is executed using the POS itself.
- If the PIN was entered and confirmed, the POS sends the PIN to the debit card for verification. If verification was successful, the requested operation is performed on the debit card. If this also is successful, the result is printed on the printer (additionally the result is send to the serial port using 9600bps with 8N1), a transaction log is stored in EEPROM and the result is shown in POS and PIN pad display.
- The POS display a message to request “Remove card!”. If the card is removed, you are back in “Insert card..” screen.

7. Reference

7.1 File System

Currently neither calls to file system functions accessing files in terminal nor in an used BasicCard are supported. This is a known limitation compared to terminal programs running on PC. Future version will remove this limitation.

7.2 Encryption Functions

Encryption functions can be used same as in PC terminal. This also applies to BasicCard commands like **StartEncryption** which are fully supported. Encryption of Compact BasicCard is not supported and will not be supported in future versions.

7.3 Supported Libraries

With BasicCard you may use libraries by including a **.def** file in your program source. Currently the BasicCard POS support only one standard library. This is the **MISC** library (misc.def). You will not get a compiler error if you use different libraries, but you will get a **PCode** error **pcNotImplemented** if you call a function of an unsupported library.

Additionally there is a special library for BasicCard POS. This library is used by including POS.def (c:\basiccardpro\pos\lib\pos.def). See reference to pos.def below.

7.4 Using Printer

The BasicCard POS has a built in printer unit. You can use this device from terminal by print statement with file number 254 or constant **Printer** defined in pos.def. E.g.:

print #Printer, “Hello World”

7.5 Using PIN Pad

An additional external PIN pad can be connected to BasicCard POS. This device is an optional feature designed to give customers an independent input/out device, e.g. for entering a PIN number. You can use this PIN pad by use of functions defined in pos.def or through file number 253 or constant PinPad (pos.def). E.g.:

print #PinPad, “Hello World”

or

line input #PinPad, Data$

7.6 Selecting Card Reader

With predefined **ComPort** variable you select the active card reader. This is same as in terminals running on PC. Inside BasicCard POS the default value of ComPort variable is 1.

Following assignment are defined for BasicCard POS built in card readers:

ComPort	Card Reader
1	Standard card reader. Located on right side of display.
2	Retailer card reader. Located on backside behind plastic cover.
3	SAM card reader. Located on backside behind security cover.

7.7 Key Pad

The BasicCard POS has a limited set of available keys. The red power on switch is only functional for switching the device on. It cannot be read through any input function. All other keys are read in by standard input functions. Typically a input must be terminated by pressing **Enter**. This is same as pressing Enter on your PC keyboard. The keys are assigned as follows:

Key / Label	Character assigned
F1	‘ ‘ (Space)
F2	‘+’
F3	‘-‘
F4	‘/’
0 .. 9	Numeric character as labeled
↓*	‘*’
↑/	‘.’
Menu	‘M’
Cancel	Whole input is discarded, ESC character with ASCII code 27 is returned.
Clear	Backspace. Previous entered character is deleted.
Enter	Enter

7.8 Real Time Clock

The BasicCard POS includes a real time clock which is running battery powered even when device is switched off. You can access this real time clock in different ways:

- The variable/function **Time$** will return an ASCII representation of current time and date.
  Following example will print the time and date using the printer
  Example: print #Printer, Time$
  See BasicCard manual for more details about Time$.
- The misc library sub routine **GetDateTime** will return the current date and time in a DateTime record. E.g. Following code will receive the current date and time:
  #include misc.def
  Public CurrentTime as DateTime
  Call GetDateTime(CurrentTime)
  Please note: GetDateTime running in POS will always return 0 in Millisecond part of DateTime type.
- The misc library function **TimeInterva**l can be used same as on PC to measure the time elapsed between two calls to GetDateTime. Because of the limitation regarding milliseconds, the precision of this is reduced to about one second.
- The misc library function **UnixTime** will work same as on PC, see BasicCard manual for more details.
- The pos library function **PosSetDateTime** will set the real time clock of the POS by use of a DateTime record. See reference to pos.def below for more details.
- The pos library function **PosSetUnixTime** will set the real time clock of the POS by use of a number (seconds since 1st January 1970). See reference to pos.def below for more details.

7.9 Using Modem and Serial Port

Modem and serial port are supported by a set of functions included in pos.def. Following functions are available (see reference to pos.def below):

- PosGetPortBaudRate
- PosSetPortBaudRate
- PosSetPortConfig
- PosSetPortConfig
- PosPortFlush
- PosPortCharCount
- PosPortLineState
- PosPortSetLine
- PosPortRead
- PosPortWrite

Additionally a limited set of file io functions can be used. For this you must use file number Modem or Serial (see pos.def). Available functions are:

- **line input**
  Line input reads input from port until a carriage return (Chr$(13)) is received. New line characters received (Chr$(10)) are ignored. Such both carriage return and new line characters are not part of received string. Using line input will generate an echo of all received characters through transmit line of port. If echo is not acceptable line input must not be used, use PosPortRead instead.
  For example:
  Public Data as String
  line input #Serial, Data
- **print**
  Print used with file number #Serial or #Modem will send data to matching port. If not suppressed by ‘;’ at end of print statement, carriage return and new line characters are appended to output data.
  For example:
  print #Serial, “Hello World”

7.10 LEDs

The BasicCard POS has built in five LEDs. These are placed below the LCD display. Four of these LEDs can be used from Basic program (one LED is the power LED and such controlled by hardware only). These LEDs are numbered 1 to 4 starting with leftmost LED. LED 1 to 3 are orange. LED 4 is green. The **PosSetLed** function defined in pos.def can be used to switch this LEDs on or off.

7.11 Unsupported Features

The RS485 port of the BasicCard POS is used by PIN Pad device. Because of this it is not supported to access the RS485 port itself.

7.12 Switching Off The Device

The BasicCard POS will switch off itself as soon as the Basic program running is ended. This means it is up to you to provide a function in your program to turn of the device. You may power off the device by ending your program through **exit** statement. If for some reason you cannot turn off the device, press the hidden reset button on backside of device.

7.13 EEPROM Variables

Eeprom variables will keep their content even if the device is powered down. Anyway all Eeprom content is lost if the same or a different application is loaded into the BasicCard POS.

7.14 BasicCard POS Library

For supporting special features of BasicCard POS there is an additional library (POS). To use this library include the file **pos.def**. E.g.:

#include pos.def

Do not forget to set a proper search path through compiler **–I** option or project parameters (Include Path). These must include path to c:\basiccardpro\pos\lib.

For a description of supported functions see following chapters.

7.14.1 Constants defined in POS.DEF

Following constants are defined in POS.def

Constant	**Value**	**Meaning**
Printer	254	File number to access printer
PinPad	253	File number to access PIN pad
Modem	252	File and port number of modem port. (Please note: Modem cannot be used with file io functions for now)
Serial	251	File and port number of serial (RS232) port. (Please note: Serial port cannot be used with file io functions for now)
Screen	0	File number to access POS Screen

Table 1 File numbers

Constant	**Value**	**Meaning**
PosF1	" " (Space)	Result when **F1** key is pressed
PosF2	"+"	Result when **F2** key is pressed
PosF3	"-"	Result when **F3** key is pressed
PosF4	"/"	Result when **F4** key is pressed
PosEnter	Chr$(13)	Result when **ENTER** key is pressed
PosCancel	Chr$(27)	Result when **CANCEL** key is pressed
PosClear	Chr$(8)	Result when **CLEAR** key is pressed
PosMenu	"M"	Result when **MENU** key is pressed

Table 2 Key codes when running in BasicCard POS

Constant	**Value**	**Meaning**
PcF1	Chr$(70)+Chr$(1)	Result when **F1** key is pressed
PcF2	Chr$(71)+Chr$(1)	Result when **F2** key is pressed
PcF3	Chr$(72)+Chr$(1)	Result when **F3** key is pressed
PcF4	Chr$(73)+Chr$(1)	Result when **F4** key is pressed
PcEnter	Chr$(13)	Result when **ENTER** key is pressed
PcCancel	Chr$(27)	Result when **ESC** key is pressed
PcClear	Chr$(8)	Result when **BACKSPACE** key is pressed
PcMenu	Chr$(74)+Chr$(1)	Result when **F5** key is pressed

Table 3 Key codes when running on Windows PC

7.14.2 RunInPos

This function allows your program to check at run time if it is running inside a BasicCard POS or e.g. on a Windows PC. For this the first command line parameter is used. In BasicCard POS Param$(1) is set to “POS”.

```
Syntax:     Function RunInPos() as Integer
```

Example:

```
#include pos.def
....
if RunInPos() then
   ‘ We are running in a real BasicCard POS
   print “Real POS“
else
   ‘ We are not running in a real BasicCard POS
   ‘ probably we are running on Windows PC
   print “Simulated POS”
end if
```

7.14.3 PosCardPowerOff

This function allows you to switch off power for selected card (as specified by ComPort variable).

```
Syntax: Sub PosCardPowerOff()
```

Example:

```
#include pos.def
....
Public Test$
ResetCard : Call CheckSW1SW2
Test$=”ABC”
Call Echo(Test$) : Call CheckSW1SW2
Call PosCardPowerOff() : Call CheckSW1SW2
....
ResetCard : Call CheckSW1SW2
Test$=”ABC”
Call Echo(Test$) : Call CheckSW1SW2
Call PosCardPowerOff() : Call CheckSW1SW2
```

7.14.4 PosPadAvailable

Check if a PIN pad is connected.

```
Syntax: Function PosPadAvailable() as Integer
```

Example:

```
#include pos.def
....
if PosPadAvailable() then
    ‘ There is a PIN pad connected
    Print #PinPad, “Hello World”
else
    ‘ There is no PIN pad connected
end if
```

7.14.5 PosPadInKey

Read a key from PIN pad without displaying it. This function returns an empty string if no key is pressed. PosPadInKey is equivalent to InKey$ statement for terminal itself.

```
Syntax: Function PosPadInKey() as String
```

Example:

```
#include pos.def
....
select case PosPadInKey()
   case “”
     ‘ No key available

   case PosCancel
     ‘ Cancel was pressed

   case PosEnter
     ‘ Enter was pressed

   case PosClear
     ‘ Clear was pressed

   case “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”
     ‘ Number was pressed

   case “.”
     ‘ Dot was pressed
end select
```

7.14.6 PosPadClearScreen

Clear PIN pad display.

```
Syntax: Sub PosPadClearScreen()
```

Example:

```
#include pos.def
....
call PosPadClearScreen()
```

7.14.7 PosPadGetXY

Get current cursor position of PIN pad.

```
Syntax: Sub PosPadGetXY(X as Integer, Y as Integer)
```

Example:

```
#include pos.def
....
Public PadX
Public PadY
call PosPadGetXY(PadX, PadY)
Print “X:” Str$(PadX) “ Y:” Str$(PadY)
```

7.14.8 PosPadSetXY

Set cursor position for PIN pad.

```
Syntax: Sub PosPadSetXY(X as Integer, Y as Integer)
```

Example:

```
#include pos.def
....
‘ Set Cursor to upper left corner
call PosPadSetXY(1, 1)
```

7.14.9 PosSetDateTime

Set date and time of POS real time clock.

```
Syntax: Sub PosSetDateTime(currenttime as DateTime)
```

Example:

```
#include pos.def
....
‘ Set date and time to 1st July 2002, 10:26 PM (22:26)
Public CurrentTime as DateTime
CurrentTime.Year=2002
CurrentTime.Month=7
CurrentTime.Day=1
CurrentTime.Hour=22
CurrentTime.Minute=26
CurrentTime.Second=0
CurrentTime.Millisecond=0
call PosSetDateTime(CurrentTime)
```

7.14.10 PosSetUnixTime

Set date and time of real time clock. This function uses Unix like representation of time and date which is seconds elapsed since 1st January 1970.

```
Syntax: Sub PosSetUnixTime(newtime as Long)
```

Example:

```
#include pos.def
....
‘ Adjust time by adding 60 seconds to old time
Public CurrentTime as Long
CurrentTime=UnixTime()
CurrentTime=CurrentTime+60
Call PosSetUnixTime(CurrentTime)
```

7.14.11 PosGetPortBaudRate

Get value of currently active baud rate for specified port.

```
Syntax: Function PosGetPortBaudRate(Port as Byte) as Long
Port: either Modem or Serial
Return value: either BaudRate or 0 if call failed
```

Example:

```
#include pos.def
....
‘ Get current baud rate of serial port
Public BaudRate as Long
BaudRate=PosGetPortBaudRate(Serial)
```

7.14.12 PosSetPortBaudRate

Set baud rate for specified port. If function fails, an error (value <>0) is returned in FileError variable.

```
Syntax: Sub PosSetPortBaudRate(Port as Byte, BaudRate as Long)
Port: either Modem or Serial
BaudRate: new BaudRate (see hardware manual for supported baud rates)
```

Example:

```
#include pos.def
....
‘ Set baud rate of serial port to 9600 baud
‘ First set FileError to 0 so we can check for error condition
FileError=0
Call PosSetPortBaudRate(Serial, 9600)
if 0<>FileError then
   ‘ An error occurred. Invalid BaudRate?
end if
```

7.14.13 PosGetPortConfig

Get port configuration string. This string includes options like number of data bits and number of stop bits. See next chapter for details about configuration string. If function fails, an error (value <>0) is returned in FileError variable.

```
Syntax: Sub PosGetPortConfig(Port as Byte, Config as String)
Port: either Modem or Serial
Config: String variable to receive configuration string
```

Example:

```
#include pos.def
....
Public PortConfig as String
‘ Get and print port configuration
‘ First set FileError to 0 so we can check for error condition
FileError=0
Call PosGetPortConfig(Serial, PortConfig)
if 0<>FileError then
   ‘ An error occurred.
else
   print #Printer, PortConfig
end if
```

7.14.14 PosSetPortConfig

Set port configuration by use of setting string. This string includes options like number of data bits and number of stop bits. If function fails, an error (value <>0) is returned in FileError variable.

Syntax: Sub PosSetPortConfig(Port as Byte, Config as String)
Port: either Modem or Serial
Config: String containing new configuration:
First 3 character must contain number of data bits, parity and number of stop bits in format “DPS”, where
D: is either **8** or **7**
P: is either **N** (or **n**) for no parity, **E** for even parity, **e** for even parity without checking parity of incoming data, **O** for odd parity or **o** for odd parity without checking parity of incoming data.
S: is either **1** or **2**
Additional optional settings may follow. Each optional setting must be separated by a colon (,). Following optional settings are possible:
XONOFF=RX -> Enable XONOFF flow control for receiver
XONOFF=TX -> Enable XONOFF flow control for transmitter
XONOFF=TX+ -> Enable XONOFF flow control for transmitter and disable transmitter when XOFF received
XONOFF=1 -> Enable XONOFF flow control for both directions
XONOFF=1+ -> Enable XONOFF flow control for both directions and disable transmitter when XOFF received
XONOFF=0 -> Disable XONOFF flow control
an **a** can be appended to all XONOFF settings. Then any character received implies XON
CTSFLOW=1 -> Enable CTS flow control (transmitter)
CTSFLOW=0 -> Disable CTS flow control (transmitter)
DSRFLOW=1 -> Enable DSR flow control (transmitter)
DSRFLOW=0 -> Disable DSR flow control (transmitter)
DCDFLOW=1 -> Enable DCD flow control (transmitter)
DCDFLOW=0 -> Disable DCD flow control (transmitter)
DTRFLOW=1 -> Enable DTR flow control (receiver)
DTRFLOW=0 -> Disable DTR flow control (receiver)
RTSFLOW=1 -> Enable RTS flow control (receiver)
RTSFLOW=0 -> Disable RTS flow control (receiver)
XONCHAR=c -> where c is a character to set XONCHAR
XOFFCHAR=c -> where c is a character to set XOFFCHAR
BREAKCHAR=c -> where c is a character to set BREAKCHAR
ERRORCHAR=c -> where c is a character to set ERRORCHAR
MODE=RX -> Enable receiver only
MODE=TX -> Enable transmitter only
MODE=1 -> Enable transmitter and receiver
MODE=0 -> Disable transmitter and receiver
ERRORMODE=R -> Replace error characters with ERRORCHAR
ERRORMODE=D -> Discard error characters
ERRORMODE=I -> Ignore errors
ERRORMODE=M -> Mark parity error (set bit 7)
BREAKMODE=R -> Replace received breaks with BREAKCHAR
BREAKMODE=I -> Ignore received breaks

Example:

```
#include pos.def
....
‘ Set serial port to 9600 baud
‘ 8 data bits, no parity, one stop bit.
‘ Use XONOFF flow control without hardware flow control.
‘ First set FileError to 0 so we can check for error condition
FileError=0
Call PosSetPortBaudRate(Serial, 9600)
Call PosSetPortConfig(Serial, _
“8N1,XONOFF=1,CTSFLOW=0,DSRFLOW=0,DTRFLOW=0,RTSFLOW=0”)
if 0<>FileError then
   ‘ An error occurred. Invalid parameter?
end if
```

7.14.15 PosPortFlush

Empty internal port buffers, by removing all received or not send characters. If function fails, an error (value <>0) is returned in FileError variable.

```
Syntax: Sub PosPortFlush(Port as Byte, Direction as String*1)
Port: either Modem or Serial
Direction: either “I” for input buffer, “O” for output buffer or “B” for
both buffers.
```

Example:

```
#include pos.def
....
‘ Flush input and output buffer
‘ First set FileError to 0 so we can check for error condition
FileError=0
Call PosPortFlush(Serial,”B”)
if 0<>FileError then
   ‘ An error occurred.
end if
```

7.14.16 PosPortCharCount

Query count of bytes available in input or output buffer or query amount of space available in input or output buffer. If function fails, an error (value <>0) is returned in FileError variable.

```
Syntax: Function PosPortCharCount(Port as Byte, QueryType as String*1) as
Integer
Port: either Modem or Serial
QueryType: either “I” for amount of characters available in input buffer,
                  “O” for amount of characters in output buffer,
                  “i” for free space remaining in input buffer
               or “o” for free space remaining in output buffer.
```

Example:

```
#include pos.def
....
Public ReceiveCount as Integer
‘ Check if characters received
‘ First set FileError to 0 so we can check for error condition
FileError=0
ReceiveCount = PosPortCharCount(Serial,”I”)
if 0<>FileError then
   ‘ An error occurred.
else
   if ReceiveCount>0 then
      ‘ There are characters available
   end if
```

```
end if
```

7.14.17 PosPortLineState

Return state of port lines. If function fails, an error (value <>0) is returned in FileError variable.

```
Syntax: Function PosPortLineState(Port as Byte, Line as String*3) as
Integer
Port: either Modem or Serial
Line: either “DSR”, “CTS”, “RI”, “DCD”, “DTR”, “RTS”
Return Value: true or false
```

Example:

```
#include pos.def
....
‘ Check if DCD is active
‘ First set FileError to 0 so we can check for error condition
FileError=0
if PosPortLineState(Serial, “DCD”) then
   ‘ DCD is active
end if
if 0<>FileError then
   ‘ An error occurred.
end if
```

7.14.18 PosPortSetLine

Set state of port lines. If function fails, an error (value <>0) is returned in FileError variable.

```
Syntax: Sub PosPortSetLine(Port as Byte, Line as String*3, NewState as
Integer)
Port: either Modem or Serial
Line: either “DTR”, “RTS”, or “BRK” (send break)
NewState: true or false
```

Example:

```
#include pos.def
....
‘ Set RTS to active state
‘ First set FileError to 0 so we can check for error condition
FileError=0
Call PosPortSetLine(Serial, “RTS”, true)
if 0<>FileError then
   ‘ An error occurred.
end if
```

7.14.19 PosPortRead

Read bytes from port. If function fails, an error (value <>0) is returned in FileError variable.

Please note: This function reads as many bytes as available at this time. It does not wait until all requested bytes are received.

```
Syntax: Function PosPortRead(Port as Byte, MaxLen as Integer) as String
Port: either Modem or Serial
MaxLen: Maximum amount of bytes to read or –1 for as much as available
Return Value: bytes read as string
```

Example:

```
#include pos.def
....
Public InputData as String
‘ Read from serial port
‘ First set FileError to 0 so we can check for error condition
FileError=0
InputData=PosPortRead(Serial, -1)
if 0<>FileError then
   ‘ An error occurred.
else
   if Len(InputData>0) then
      ‘ Data was received
   end if
end if
```

7.14.20 PosPortWrite

Write bytes to port. If function fails, an error (value <>0) is returned in FileError variable.

Please note: This function may fail if output buffer is full.

```
Syntax: Sub PosPortWrite(Port as Byte, Data as String)
Port: either Modem or Serial
Data: data to write
```

Example:

```
#include pos.def
....
‘ Write to serial port
‘ First set FileError to 0 so we can check for error condition
FileError=0
Call PosPortWrite(Serial, “Hello World”)
if 0<>FileError then
   ‘ An error occurred.
end if
```

7.14.21 PosLedSet

Switch LED on or off.

```
Syntax: Sub PosLedSet(LEDNum as Byte, LEDVal as Integer)
LEDNum: number of LED to change1 to 4
LEDVal: true or false
```

Example:

```
#include pos.def
....
‘ Switch LED 1 on
Call PosLedSet(1, true)
‘ Switch LED 1 off
Call PosLedSet(1, false)
```