# EmpiFis for Empirija FMB

FMB's Command Reference

Release 2.3 January 13, 2011

UAB Empirija

Savanorių pr. 271-342, Kaunas, Lietuva

Phone: +370 37 313395

E-Mail: empirija@empirija.lt

WWW: www.empirija.lt

# Table of contents

# EmpiFis for Empirija FMB overview

## What is "EmpiFis for Empirija FMB"

EmpiFis for Empirija FMB is an interface that allows Empirija FMB[1] hardware to be easily integrated into POS systems based on Microsoft Windows 95/ 98/ ME, Microsoft Windows NT/ 2000/ XP and Linux. This interface is implemented in three ways:

- DLL exported functions "empifis.dll"
- In-process automation server "empifisx.dll"

## Who should read this document

The Programmer's Guide is targeted to an application developer who requires access to Empirija FMB.

This guide assumes that the reader understands the following:

- The POS terminology and working principles.
- Development environments, such as Microsoft Visual Basic, Microsoft Visual C++, Borland C++ Builder , Borland Delphi.

[1] Fiscal Memory Block

## What's new

EmpiFis version 2.3 is dedicated for Empirija FMB version 21.00 or later. This version of EmpiFis has same list of commands as earlier versions of EmpiFis. **But because of changes in internal communication protocol with FMB, this version of EmpiFis isn't backward compatible with earlier versions Empirija FMB.** If this version of EmpiFis will be used with earlier versions of the FMB then commands will usually return ERR_RECEIVE error code.

| Version | Commands | Comments |
|---|---|---|
| 1.4 | SetPrinterType | Added support for additional types of printers |
| | SetFooter | Receipt can have additional 4 lines for footer |
| | EnableFooter | Enables or disables printing of footer lines |
| | SetVat | Expanded VAT's range for 4 to 6 |
| | SetCurrency | Supports receipt settlement in four currencies (local, € and 2 user defined) |
| | MoneyInCurr | Enclose cash (of currency) to drawer. Replaces MoneyIn command |
| | MoneyOutCurr | Withdraw cash (of currency) to drawer. Replaces MoneyOut command |
| | GetFiscalInfo | FMB now can return wider range of information |
| | CustomerDisplay2 | Replaces CustomerDisplay command |
| | PrintBarCode | Barcodes can be printed on receipt (only on some types of printers) |
| | PrintDepositReceive, PrintDepositRefund | Print received or refunded deposit on non-fiscal receipt |
| | PrintTareItem, PrinTareItemVoid | Print received tare on non-fiscal receipt |
| | EndFiscalReceiptCurr, GoodsReturnCurr | Supports receipt settlement in four currencies (local, € and 2 user defined). These commands replaces EndFiscalReceipt and GoodsReturn |
| 2.0 | GetFiscalInfo | Additional information about e-journal |
| | ClearMemoryDate | Clearing FMB memory with date parameter. This command replaces ClearMemory (with FMB number parameter) |
| | FiscalizationVAT | Fiscalizes FMB with additional company VAT code parameter. This command Fiscalization. |
| | SetCompanyVATCode | Sets company VAT code. Initial set is done by FiscalizationVAT command |
| | PrintSumPeriodicReportByNumber, PrintPeriodicReportByNumber | Print fiscal reports from selected internal of Z reports. |
| | FinalizeZReport | Retrieves e-journal from FMB and clears FMB e-journal storage. This command is part of PrintZReport command |
| | GetFiscalInfo2 | Command is similar to GetFiscalInfo except information is returned as function result instead or error code. **Implemented in** |

| Version | Commands | Comments |
|---|---|---|
| | | **ActiveX only** |
| | COMConnect | Opens required serial ports. **Implemented in ActiveX only** |
| | COMDisconnect | Closes used serial ports. **Implemented in ActiveX only** |
| 2.2 | PrintCopyOfReceipt | Allows to print copy of any receipt from current e-journal |
| | GetCopyOfReceipt | Retrieves copy of any receipt from current e-journal |
| | FormatCard | Formats internal FMB memory card. This card is used as storage for e-journal files |
| | PrintCardDirectory | Prints list of files on internal FMB memory card. |
| | GetFiscalInfo | Changed information parameters for e-journal |
| 2.3 | ReceiptSuspend | Increases FMB's counters for suspended receipts |
| | ReceiptRecall | Decreases FMB's counter for suspended receipts |
| | GetFiscalInfo | Additional parameter to check paper state |

# EmpiFis for Empirija FMB model

The FMB is external device which controls receipt printer, customer display and cash drawer. FMB is an intellectual device with own processor, memory and internal software. It receives commands from control device (usually computer or POS), executes them and returns the result code about command's execution.

## Methodology of the FMB calculations

## Restriction

For item price and quantity representation is used numbers with 10 significant digits. Decimal point position isn't fixed.

Product of item price and quantity can't exceed maximum value of 99999.99.

Totals in the FMB are stored in smallest available coin "cent". Cent is equal to hundredth of base unit of local currency "Litas". Total amount of receipt can't exceed maximum value of 4294967295 cents (total amount is stored in unsigned 32-bits integer). Otherwise the FMB will raise an error, and incorrect total will be printed on receipt.

Total turnovers in Z report are restricted to maximum value of 4294967295 cents.

### Rounding

Total amount for item always is rounded to two decimal digits (to smallest available coin "cent"). Rounding rules are such:

If a third decimal digit is less than five then second digit will be left as is.

If a third decimal digit is equal or more than five then second digit will be incremented by one.

Example:

1. $10.2366 \approx 10.24$
2. $25.2346 \approx 25.23$
3. $34.2950 \approx 34.30$

### Methodology of the FMB calculations

Base unit of the FMB calculations is receipt.

#### *Totals by VAT within receipt*

Product of item price and quantity is rounded to two decimal digits. This amount is added to corresponding pile. There are different piles for totals distributed by item VAT. Receipt total is equal to sum of these totals. Count of piles is equal to count of records in FMB's VAT table. This version of FMB has six VAT records.

Discount or surcharge for item decrements or increments amount of corresponding pile.

(0.1)

$Total_{VATx} =$

$\sum_{VATx} Round(Price_{Item} \times Quantity_{Item}) + \sum_{VATx} Round(Amout_{Discount/Surcharge}), x \in A..F$

(0.2) $Total_{Receipt} = \sum_{i=A}^{F} Total_{VATi}$

Where "A..F" corresponds to numbers of VAT.

### *VAT amounts by VAT within receipt*

VAT's amounts are calculated on end of receipt for each total pile.

(0.3) $VATAmount_{VATx} = Round\left(Total_{VATx} \times \frac{VATValue_{VATx}}{100+VATValue_{VATx}}\right), x \in A..F$

Where "VATValue" contain VAT rate in percent.

### *Payment in non-local currency*

If payment for receipt was accomplished by non-local currency then amount in local currency will be calculated by using formula:

(0.4) $Amount_{LocalCurr} = \sum_{i=1}^{3} Round(Rate_{Curr\ i} \times Amount_{Curr\ i})$

### *Totals between Z reports*

Turnovers in Z report will be equal to sum of corresponding values that was calculated in receipts.

(0.5) $TotalTurnover_{VATx} = \sum_{Receipts} Total_{VATx}, x \in A..F$

(0.6) $TotalTurnover = \sum_{i=A}^{F} TotalTurnover_{VATi}$

And VAT's amounts in Z report will be equal to sum of receipt VAT's amounts.

(0.7) $TotalVATTurnover_{VATx} = \sum_{Receipts} Total_{VATx}, x \in A..F$

## State model

The FMB has 4 **states**. **IDLE**, **ZIDLE**, **FIS, NONFIS** and **HARD**. Each state describes what kind of commands can be carried out and what kind of procedure is started (fiscal receipt, non-fiscal receipt, etc.). The FMB switches states by itself depending on performed command.

The FMB will switch into **ZIDLE state** when Z report was performed. In this state can be performed commands for setting up the FMB parameters - programming of VAT table, change the FMB time or date and etc.

The FMB will switch into **IDLE state** when fiscal receipt was printed. In this state you can start new fiscal or not fiscal receipts, open cash drawer, and output information to customer display.

When fiscal receipt is started, the FMB will switch into **FIS state**. In this state can be performed fiscal receipt commands such as sell item, return item, apply discounts/additions for item or receipt and etc.

When not fiscal receipt is started, the FMB will switch into **NONFIS state**. In this state can be performed not fiscal receipt command such as print not fiscal receipt line or finish not fiscal receipt.

The FMB **state** can be checked any time.

There is a special FMB state **HARD**. The **HARD state** indicates an errors condition with the FMB or connected to the FMB peripherals such as receipt printer or customer display. When the FMB switches to this state its work will be halted.

The FMB functions flowchart and state model is showed in **Figure 1** Fiscal Memory Block Flowchart and States**.**

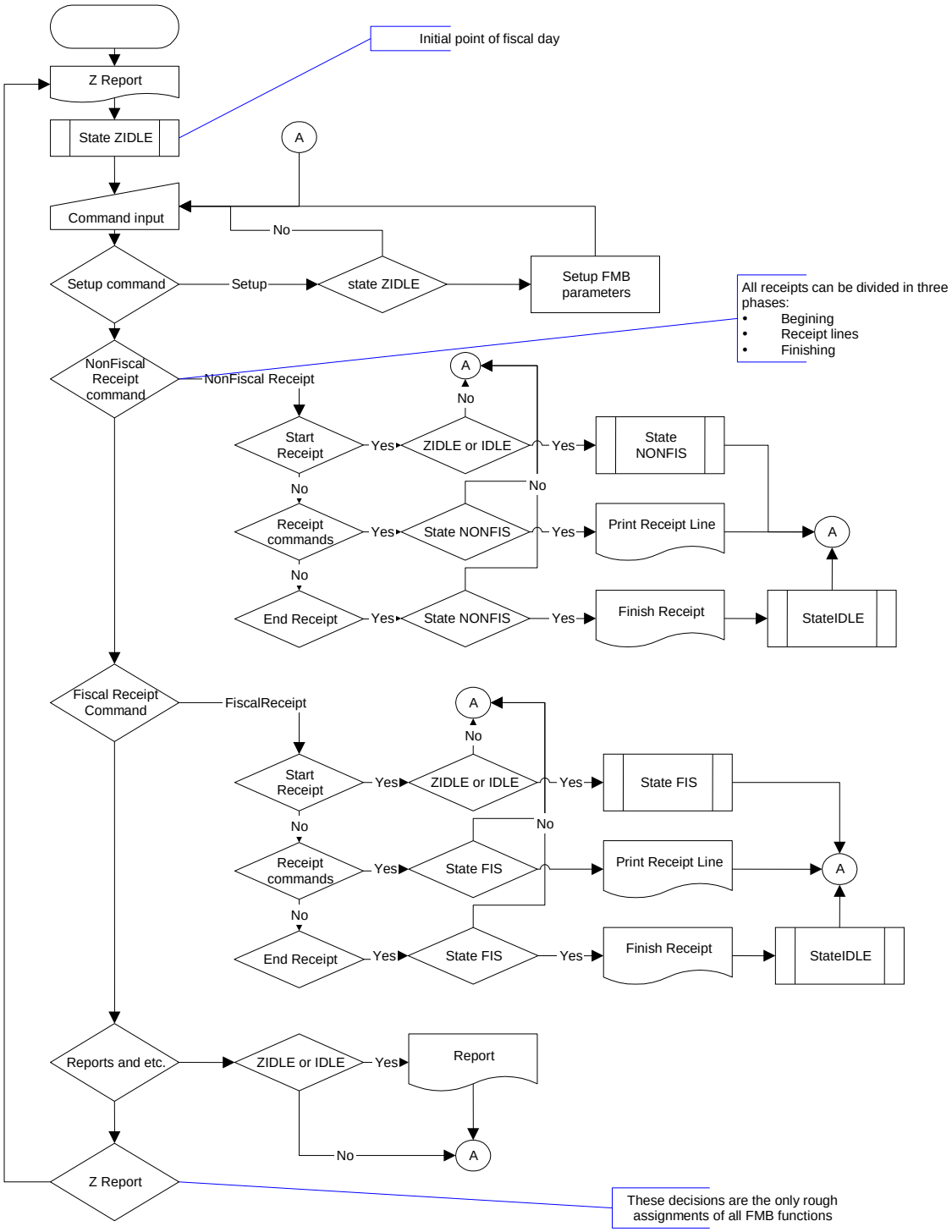


**Figure 1 Fiscal Memory Block Flowchart and States**

## Result Code model

The rules of the result code model are as follows:

- Every function or method returns a **result code.**
- The **result code** FMB_OK is assigned the value of zero. Non-zero values indicate an error.
- The full listing of **result codes** can be founded in Error! Reference source not found..

## Hypothetical workday model

There is description of hypothetical workday with the FMB.

First of all preparation to work:

1. X or Mini X report is printed by issuing **PrintXReport** or **PrintMiniXReport** command to make sure the FMB turnover is equal zero, or in other words – the FMB is in ZIDLE state. This isn't mandatory step; it's only preventing improper start of workday.

2. Cash is enclosed by **MoneyInCurr** command into the cash drawer. The cash that was withdrawn on last workday end should be enclosed back into cash drawer. Please note that even money were left in cash drawer, the FMB will clear cash counters after printing of Z report. From the FMB point of view if there wasn't no enclosure then can be encountered problem with change for receipt – can be raised error ERR_DEFICIENT_CASH_DRAWER.

Work with the FMB. There are two types of receipts that can be printed by the FMB – fiscal and non-fiscal.

Steps to format a fiscal receipt:

1. **BeginFiscalReceipt** command to start fiscal receipt is issued,

2. Additional comment lines are printed by **PrintCommentLine** command to expand receipt header,

3. Receipt item is printed by **PrintRecItem** command. There can be performed additional steps for this subsection:

   **3.1. Additional comment lines are printed to describe item. Comments can be printed either before or after item,**

   3.2. An addition or discount for item can be applied by **DiscountAdditionForItem** command. Can be applied sub sequential discount/addition operations, and these operation can be commented by additional comment lines.

4. If there is an item that customer want to decline by any reason and this item is already send to the FMB then **ItemReturn** command must be issued.

5. An addition or discount for receipt can be applied by **DiscountAdditionForReceipt** command. Can be applied sub sequential discount/addition operations, and these operation can be commented by additional comment lines. After this command can be printed additional items (item 3 of this list).

6. Command to finish Fiscal receipt is issued. After this command can't be printed any additional information on receipt. There is to possibilities to finish fiscal receipt:

   6.1. If this receipt is for normal sale then should be issued **EndFiscalReceiptCurr** command.

   6.2. If this receipt is for goods returns then should be issued **GoodsReturnCurr** command

Steps to format non-fiscal receipt:

1. **BeginNonFiscalReceipt** command to start non-fiscal receipt is issued,
2. Receipt items are printed by **PrintNonFiscalLine** command,
3. For deposit operations commands **PrintDepositReceive** or **PrintDepositRefund** are used,
4. For tare operations commands **PrintTare** and **PrintTareItemVoid** are used.
5. Issuing **EndNonFiscalReceipt** command closes receipt.

Without these two types of receipts also cab can be printed any reports, except Z report. Also cash can be enclosed by **MoneyInCurr** or withdrawn by **MoneyOutCurr** commands due workday. These procedures are performed in **IDLE state** i.e. when fiscal or non-fiscal receipt isn't started.

To end workday are performed these steps:

1. X report is printed,
2. Cash in amount shown at X report is withdrawn from cash drawer.
3. Z report is printed by **PrintZReport** command.

The flowchart for this hypothetical day is shown in **Figure 2** Flowchart of hypothetical workday**.**

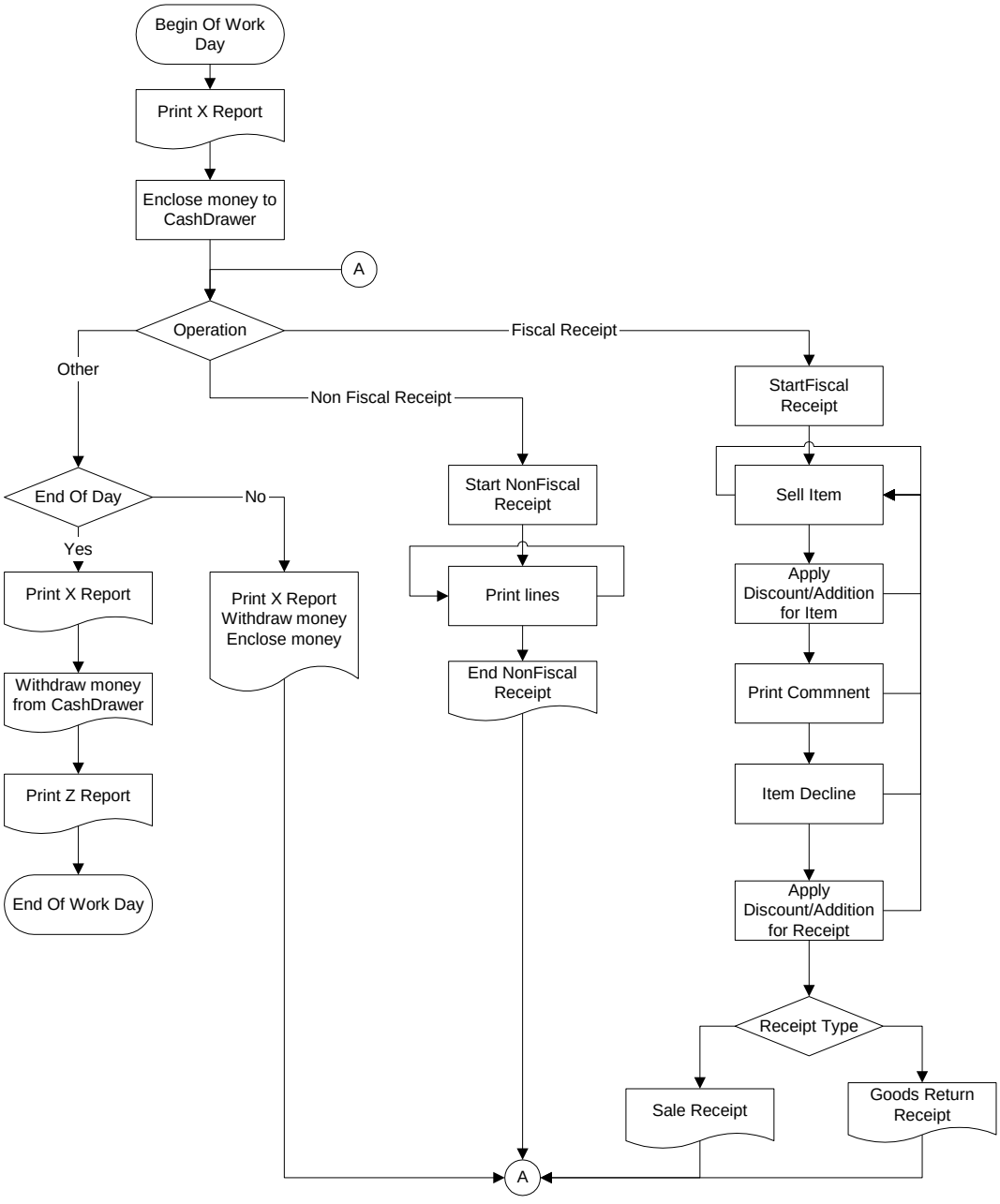


**Figure 2 Flowchart of hypothetical workday**

## Electronic Journal model

Each receipt printed by FMB is stored to FMB electronic journal. According local law, information written to electronic journal must be equal to information that is printed by printer. There can be minor differences such as printer can print lines in bold, while file will contain only text information.

POS program should take care of retrieval of FMB e-journal. There are several legal requirements for e-journal:

1. E-journal can be cleared from FMB only when it was successfully retrieved and stored to permanent storage;
2. E- journal must be simple text file, which should be opened with any available text editor;
3. There must be at least two places of permanent storage, and one of them must be external one;
4. E-journal file name must be made from e-journal retrieval date from 8 digits YYYYMMDD.

Empirija suggest some tips for e-journal:

1. FMB contains SD memory card that is used as permanent storage for e-journal. Due restriction of used file system on SD card there can be stored 512 e-journal files;
2. It is strongly recommended to set read-only attributes on stored e-journal files to avoid accidental change on content;
3. E-journal file name could be made with some additional information such as "YYYYMMDD_FMBNumber_ZNumber_CRC32.txt"
   Where:YYYYMMDD – is date of printing of Z report,
   FMBNumber – Number of FMB that printed Z report,
   ZNumber – Number of Z report, CRC32 – control sum of this e-journal.
   E.g. file name could look: "20060810_2001021700_25_10BEDF.txt".
   This could provide more information about stored e-journal, and doesn't contradict with legal requirement.
4. E-journal can be retrieved only before storing it to permanent storage or after each printed receipt. We suggest retrieving it after printing of each receipt to avoid longer delay on the end of the day.

Hypothetical e-journal retrieval algorithm for workday is shown in **Figure 3 Recommended e-journal retrieval algorithm**. Suggested detailed algorithm of e-journal retrieval is shown in **Figure 4 Detailed algorithm of FMB e-journal retrieval**

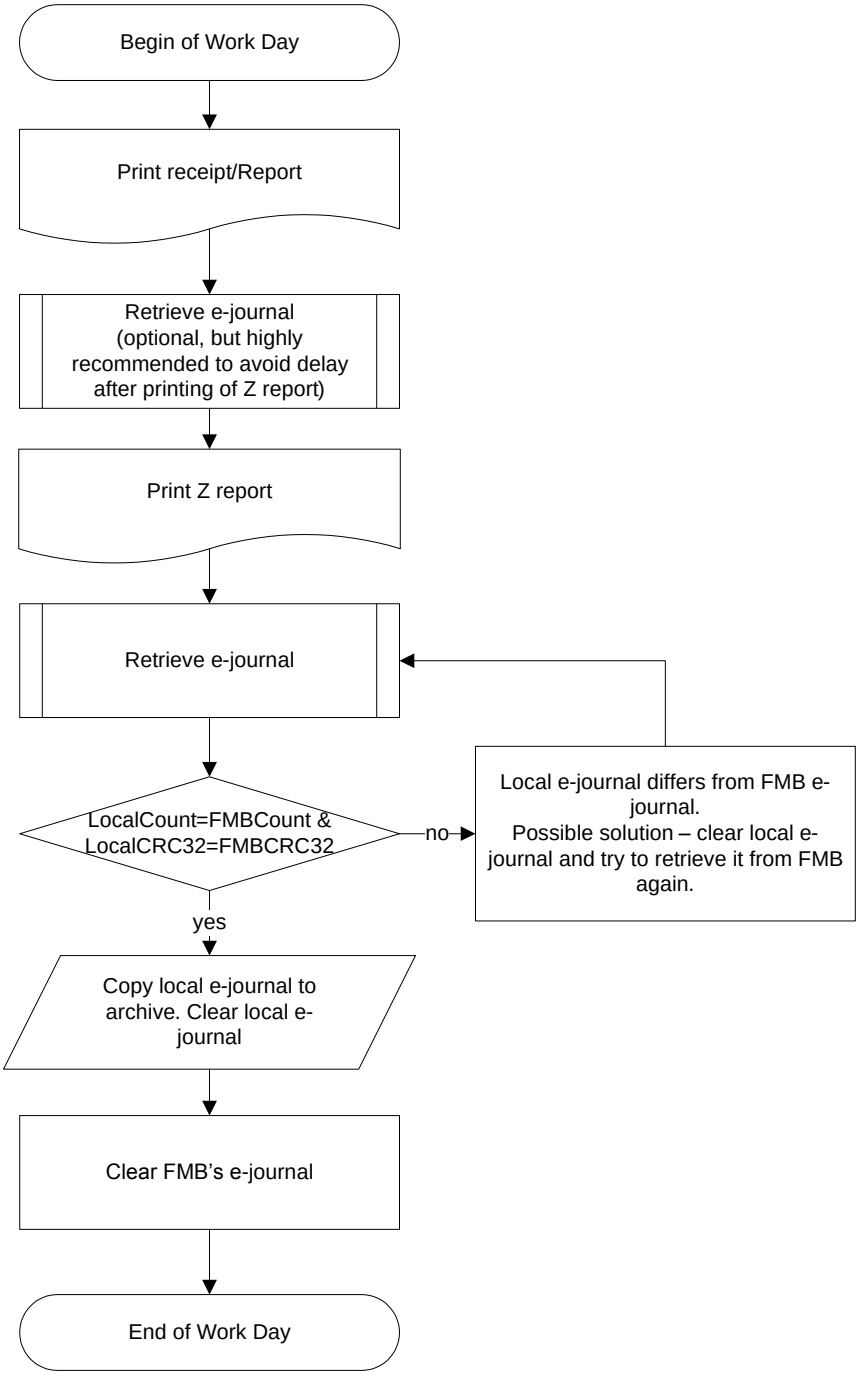


**Figure 3 Recommended e-journal retrieval algorithm**

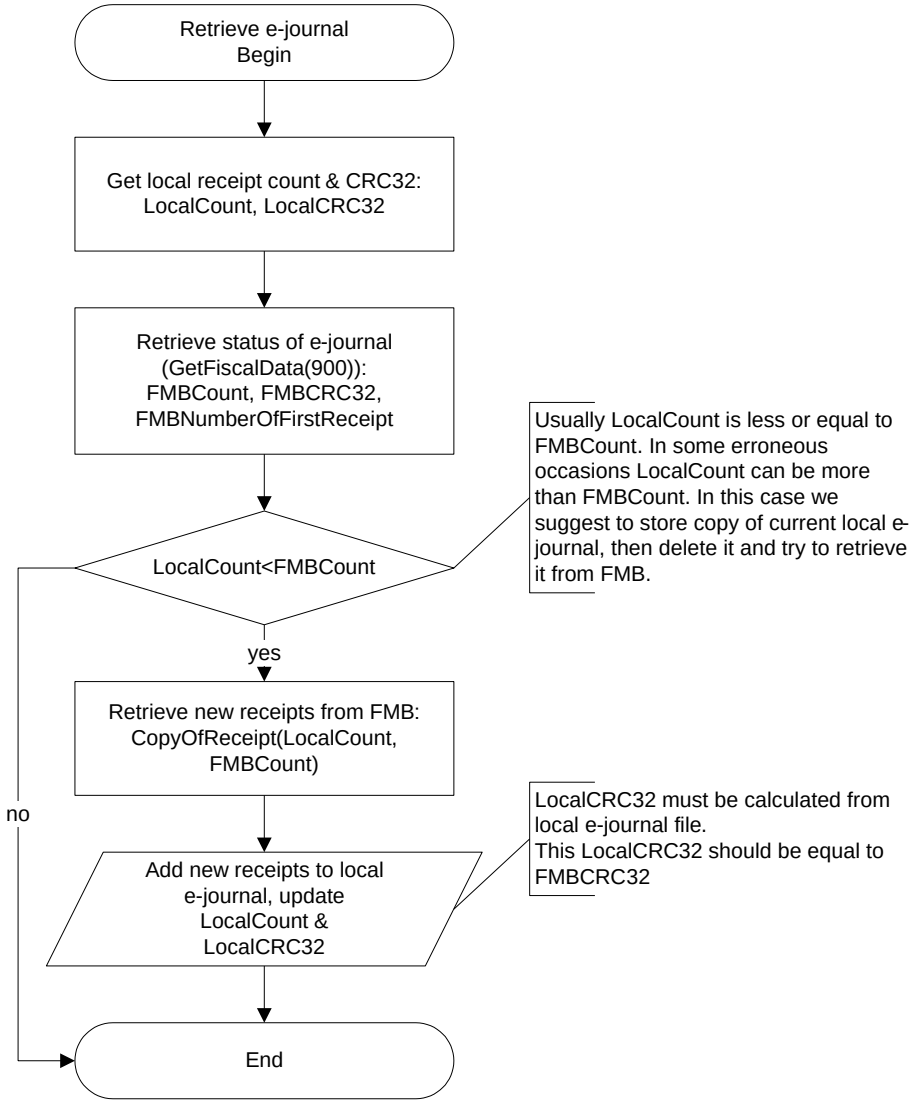


**Figure 4 Detailed algorithm of FMB e-journal retrieval**

## EmpiFis functions

There are presented short description of implemented functions in the Empifis libraries. Each function described in such way:

- State when function is available. Function can be send to FMB in any state, but they will be executed only when meets required state,
- Syntax[2] [3] of function implemented in EmpiFis.DLL library,
- Description of the function, parameters and main notes.

There are descriptions of used variable types:

- **Integer** - represents a subset of the whole numbers. Format is signed 32-bit,
- **Double –** defines a set of numbers that can be represented with floating-point notation. Format size is 8 bytes,
- **PChar** - a pointer to a Char (that is, in its current implementation, to an AnsiChar). These character pointers are used to manipulate null-terminated strings,
- **WideString** - represents a dynamically allocated string of 16-bit Unicode characters. WideString is compatible with the COM BSTR type.

[2] All syntaxes are served in Pascal language

[3] Delphi unit of declaration of exported EmpiFis functions is included in **Appendix**

## System function's list

### Reset Fiscal

Available in state: ANY.

Syntax DLL: ResetFiscal (): Integer

Syntax OLE: ResetFiscal (): Integer

Remark: Resets the FMB: switch state **IDLE.** The started receipt will be aborted. This command didn't work if the FMB state is **HARD.** In most cases this command is used to cancel started fiscal or non-fiscal receipt.

### ClearMemoryDate

Available in state: IDLE, ZIDLE.

Syntax DLL: ClearMemoryDate (pstrDate: PChar): Integer

Syntax OLE: ClearMemoryDate (wstrDate: WideString): Integer

Remark: Clears the memory of the FMB. **This command can be executed only with responsible technical person and you MUST have consent from Tax inspection.**
Variable pstrDate contains current date in format "YYYY.MM.DD" where YYYY – year, MM – month, DD – day. If month or day is less than 10 then fill spaces with '0' character**.**

### FiscalizationVAT

Available in state: IDLE, ZIDLE.

Syntax DLL: FiscalizationVAT (pstrDate,
pstrCompanyVATCode: PChar): Integer

Syntax OLE: FiscalizationVAT (wstrDate,
wstrCompanyVATCode: WideString): Integer

Remark: This command turns the FMB to fully operated state id est. **after executing this command all fiscal receipts data will be stored in fiscal memory, fiscal logotype will be printed on each fiscal receipt and Z report. Also will be available the all FMB reports.** This command can't be reverted.
Variable pstrDate contains current date in format "YYYY.MM.DD" where YYYY – year, MM – month, DD – day. If month or day is less than 10 then fill spaces with '0' character.
Variable pstrCompanyVATCode – contains company VAT code of FMB's owner.

## Set-ups function's list

### SetPrinterType

Available in state: IDLE, ZIDLE.

Syntax DLL: SetPrinterType (iPrinterType: Integer): Integer

Syntax OLE: SetPrinterType (iPrinterType: Integer): Integer

Remark: Sets printer type connected to the FMB.
Variable iPrinterType indicates type of printer and can contain values. Usually there is no need to use this function as FMB automaticaly detects printer type.
0 - Epson 210
1 - Orient
2 - Orient (80)
3 - TM-T88IV
4 - CHD TH582/TH200
5 - Epson 300

### SetCompressionMode

Available in state: IDLE, ZIDLE.

Syntax DLL: SetCompressionMode (iCompression: Integer): Integer

Syntax OLE: SetCompressionMode (iCompression: Integer): Integer

Remark: This command allows setting spacing between printed lines on receipt. This particularly can be useful on long receipts. In this case if spacing is small – receipt will be shorter. In other hand larger spacing makes receipt more readable.
Variable iCompression can contain value in range between 15 and 24.

### SetTillImpulse

Available in state: IDLE, ZIDLE.

Syntax DLL: SetTillImpulse (iTillImpulse1, iTillImpulse2: Integer): Integer

Syntax OLE: SetTillImpulse (iTillImpulse1, iTillImpulse2: Integer): Integer

Remark: The FMB controls the cash drawer also. This command sets up duration of till open impulse. Duration of impulse varies on cash drawer, and is determined experimentally.
Variables iTillImpulse1 and iTillImpulse2 values should be used in range from 0 to 255. They define pulse (on time = iTillImpulse1 x 10ms / off time = iTillImpulse2 x 10ms) to the connector pin.

### SetDate

Available in state: ZIDLE.

Syntax DLL: SetDate (pstrDate: PChar): Integer

Syntax OLE: SetDate (wstrDate: WideString): Integer

Remark: Sets the date of FMB. The prior date can't be set. **If there was made a mistake and was set wrong date, should be contacted to Empirija's service.**
Variable pstrDate contains new date in format "YYYY.MM.DD" where YYYY - year, MM - month, DD - day. If month or day is less than 10 then fill spaces with "0" character**.**

### SetTime

Available in state: ZIDLE.

Syntax DLL: SetTime (pstrTime: PChar): Integer

Syntax OLE: SetTime (wstrTime: WideString): Integer

Remark: Sets the FMB time.
Variable pstrTime contains new time in format "HH.MM" where HH - hours and MM - minutes. If value of hours or minutes is less than 10 then fill spaces with "0" character.

### SetHeader

Available in state: IDLE, ZIDLE.

Syntax DLL: SetHeader (pstrLine1, pstrLine2,
pstrLine3, pstrLine4: PChar): Integer

Syntax OLE: SetHeader (wstrLine1, wstrLine2,
wstrLine3, wstrLine4: WideString): Integer

Remark: There are four lines available to print in beginning of receipt. Fiscal receipt according local law must contain information about seller: company name, address, and VAT payer number.
Variables pstrLine1, pstrLine2, pstrLine3 and pstrLine4 are null terminated strings, and can't exceed 40 characters in length.

### SetFooter

Available in state: ANY.

Syntax DLL: SetFooter (iAttrib1: Integer; pstrLine1: PChar;
iAttrib2: Integer; pstrLine2: PChar;
iAttrib3: Integer; pstrLine3: PChar;
iAttrib4: Integer; pstrLine4: PChar): Integer

Syntax OLE: SetFooter (iAttrib1: Integer; wstrLine1: WideString;
iAttrib2: Integer; wstrLine2: WideString;
iAttrib3: Integer; wstrLine3: WideString;
iAttrib4: Integer; wstrLine4: WideString): Integer

Remark: There can be printed up to four lines at the end of receipt. These lines will be printed between totals and fiscal logotype. Each of these lines can have own attributes.
Variables pstrLine1, pstrLine2, pstrLine3 and pstrLine4 are null terminated strings, and can't exceed 40 characters in length.
Variables iAttrib1, iAttrib2, iAttrib3 and iAttrib4 contain attributes for corresponding line. Attribute must define font size and/or additional properties. **Attributes for footer lines should be used with caution, because of them line can exceed paper width and will be automatically wrapped.**
The font ID and properties are combined by binary OR function. Available codes are showed in the table below.

| Value (HEX) | Description |
|---|---|
| 0x40<br>0x41 | Font size 9x9<br>Font size 7x9 |
| 0x44 | Underline |
| 0x48 | Bold |
| 0x50 | Double height |
| 0x60 | Double width |

Footer set-up automatically enables its printing on receipt. The **EnableFooter** command disables or enables footer printing.

### EnableFooter

Available in state: ANY.

Syntax DLL: EnableFooter (iAction: Integer): Integer

Syntax OLE: EnableFooter (iAction: Integer): Integer

Remark: This command enables or disables printing of programmable lines of receipt's footer.
Variable iAction contains 0 or 1: 0 - disables printing, 1 - enables printing.
Receipt's footer printing will be automatically enabled after setup of footer lines by **SetFooter** command.

### SetVat

Available in state: ZIDLE.

Syntax DLL: SetVat (iVATID: Integer; rVATValue: Double): Integer

Syntax OLE: SetVat (iVATID: Integer; rVATValue: Double): Integer

Remark: The FMB has VAT table for six tax rates.
Variable iVATID is identification number of VAT to be changed. Valid range of iVATID is from 0 to 5 (VAT ID corresponds to letters from A to F).User actually can't set the fourth tax. **The VAT number 3 can't be changed and its value permanently set to 0. This vat used for Tax-Free item sale.** This tax-free VAT identification number is kept

for backward compatibility with earlier developed POS programs.
Earlier versions of the FMB support only four rates of VAT.
Variable rVATValue contains new VAT value.

### SetCurrency

Available in state: ZIDLE.

Syntax DLL: SetCurrency (iCurrencyID: Integer;
pstrDescription: PChar; rRate: Double): Integer

Syntax OLE: SetCurrency (iCurrencyID: Integer;
wstrDescription: WideString; rRate: Double): Integer

Remark: The FMB can accept several currencies as payment in cash. There are four slots in currency.
Valid range of value of variable iCurrencyID is from 0 to 3. **The currency with number 0 can't be changed. This ID is used for the national currency. The currency with index number 1 is dedicated to € and only rate can be changed.**
Variable pstrDescription contains abbreviation of currency. This abbreviation can't exceed 3 characters in length.
Variable rRate contains currency rate. **If rate is set equal to zero then this currency will be disabled.** .

### AllowGoodsReturn

Available in state: IDLE, ZIDLE.

Syntax DLL: AllowGoodsReturn (pstrFMBNumber: PChar): Integer

Syntax OLE: AllowGoodsReturn (wstrFMBNumber: WideString): Integer

Remark: This command was used to enable or disable refund receipt command **GoodsReturnCurr** in the FMB. **Receipt refund command is always enabled nn current version of FMB**.
Variable pstrFMBNumber contains the number of the FMB.

### SetCompanyVATCode

Available in state: ZIDLE.

Syntax DLL: SetTime(pstrCompanyVATCode: PChar): Integer

Syntax OLE: SetTime(wstrCompanyVATCode: WideString): Integer

Remark: This command is used to set company VAT code. This code initially is set durind fiscalization time.
Variable pstrCompanyVATCode – contains company VAT code of FMB's owner.

## Information function's list

### GetFiscalInfo

Available in state: ANY.

Syntax DLL: GetFiscalInfo (iInfoType: Integer; var pstrResult: PChar): Integer

Syntax OLE: GetFiscalInfo (iInfoType: Integer; var wstrResult: WideString): Integer

Remark: This command is used to retrieve some information from the FMB: version, current state, receipts numbers etc. As result of execution this command the FMB return requested data in addition to error code. Data is returned as string. Retrieved data is returned in variable pstrResult.
Variable iInfoType contains identificator of required data:

0 - Total of started receipt;
1 - Day's turnover;
2 - Total number of the next receipt;
3 - Number of the FMB;
4 - Version of the FMB's software;
5 - FMB's current date;
6 - FMB's current time;
7 - Count of the Z reports;
8 - Number of the next day's receipt;
9 - State of the FMB;
IDLE = '0';
NONFIS = '1';
FIS = '2';
HARD = '3';
ZIDLE = '4';
10 - Receipt header. All four lines are returned jointly. Received string will be such:
<Line1><FS>[4]<Line2><FS><Line3><FS><Line4>;
11 - Receipt footer. Similar to receipt header – all four lines and its attributes are returned jointly:
<Attrib1><FS><Line1><FS><Attrib2><FS><Line2><FS><Attrib3><FS><Line3><FS><Attrib4><FS><Line4>;
30..33 - Currency table item with number corresponded number (0..3). It will be returned in same way as it set. Received string will be such:
<CurrencyID><FS><Description><FS><Rate>;

[4] Abbreviation of FS stands for field separator. Its value is equal to $1C in hexadecimal.

40..45 - VAT table item with number corresponded number (0..5). It will be returned in same way as it set. Received string will be such: <VATID><FS><VATRate>;

50..53 - Amount of currency with corresponded number (0..3) in till. Amount can be requested only for programmed currencies. Otherwise function returns an error. Received string will be such: <CurrencyID><FS><Amount>;

With this function can be retrieved full content of X report. Further are list of constants that are used to retrieve these values.

70..75 - Total amount by VAT. Received string will be such: <VATID><FS><TotalAmount>;

80..85 - VAT's total amount by VAT. Received string will be such: <VATID><FS><VATAmount>;

90..93 - Amount of enclosed cash by currencies. Received string will be such:
<CurrencyID><FS><Amount>;

100..103 - Amount of withdrawn cash by currencies. Received string will be such:
<CurrencyID><FS><Amount>;

110..115 - Return's total amount by VAT. Received string will be such: <VATID><FS><TotalAmount>;

120..125 - Return's VAT's total amount by VAT. Received string will be such:
<VATID><FS><VATAmount>;

130 - Amount of sale in cash.

140..143 - Amount of sale in credit by credits. Received string will be such:
<CreditID><FS><Amount>;

150..155 - Total of discounts by VAT. Received string will be such: <VATID><FS><Amount>;

160..165 - Total of surcharges by VAT. Received string will be such: <VATID><FS><Amount>;

170 - Date of VAT's change. Format of date is YYYYMMDD;

171 - Date of last clearing. Format of date is YYYYMMDD;

180..183 - Return's total amount by credit. Received string will be such:
<CreditID><FS><Amount>;

190 - Total amount of tare;

191 - Quantity of tare;

192 - Return's receipts counter;

202 - Grant total;

203 - Counter of VAT's change;

204 - Counter of clearing;

205 – Paper roll sensor status.
18 – OK (2+16);

30 – Paper near end (2+12+16);
126 – Paper not present (2+12+16+96);

Decimal number indicates printer's paper roll status byte. Paper roll sensor status byte:

| Bit | On/Off | Hex | Dec | Remark |
|---|---|---|---|---|
| 0 | Off | 00 | 0 | Not used. Fixed to Off. |
| 1 | On | 02 | 2 | Not used. Fixed to On. |
| 2, 3 | Off | 00 | 0 | Paper roll near-end sensor: paper adequate. |
| | On | 0C | 12 | Paper roll near-end sensor: paper near end. |
| 4 | On | 10 | 16 | Not used. Fixed to On. |
| 5, 6 | Off | 00 | 0 | Paper roll end sensor: paper present. |
| | On | 60 | 96 | Paper roll end sensor: paper not present. |
| 7 | Off | 00 | 0 | Not used. Fixed to Off. |

There are list of additional parameters which are used to retrieve e-journal from FMB's internal memory:

900 – last index on receipt in FMB's e-journal, current CRC32 and number of first printed receipt in e-journal. CRC32 is returned as decimal number. If last index is equal 0 then e-journal is empty. Received string will be such:
< ID ><FS>< CRC32><FS><NumberOfReceipt>;

### GetFiscalData

Available in state: ANY.

Syntax DLL: GetFiscalData(iInfoType: Integer;var pstrResult: String10[5]): Integer

Syntax OLE: -

Remark: **This function is kept for backward compatibility with earlier developed POS programs**. Function use isn't recommended with reason to eliminate non-standard variable type of String10.
Due restriction of type of second parameter, this command can mangle returned information if „iInfoType"isn't in range of 0 till 9.
The **GetFiscalInfo** function should be used instead this one.

[5] String10 is custom variable type. It is equal to string [10] (string of 10 characters)

## Money and Drawer function's list

### MoneyInCurr

Available in state: IDLE, ZIDLE.

Syntax DLL: MoneyInCurr (iCurrencyID: Integer; rAmount: Double): Integer

Syntax OLE: MoneyInCurr (iCurrencyID: Integer; rAmount: Double): Integer

Remark: This command used to enclose additional cash to the cash drawer. After performing **PrintZReport** command cash counters are zeroed, and if cash wasn't withdrawal from then these counters must be restored to previous values.
Variable iCurrencyID contains identification number of currency to enclose. As mentioned in **SetCurrency**, there are 4 currencies that can be used in the FMB. The currency with number 0 corresponds to local one.
Variable rAmount contains amount of cash.

### MoneyOutCurr

Available in state: IDLE, ZIDLE.

Syntax DLL: MoneyOutCurr (iCurrencyID: Integer; rAmount: Double): Integer

Syntax OLE: MoneyOutCurr (iCurrencyID: Integer; rAmount: Double): Integer

Remark: This command used to withdraw cash to the cash drawer. Command can be performed any time of day: on fixed hours, amount in cash drawer exceed defined amount etc. Usually this operation is performed on the end of day before Z report.
Variable iCurrencyID contains identification number of currency to withdraw.
Variable rAmount contains amount of cash.

### MoneyOutCurr2

Available in state: IDLE, ZIDLE.

Syntax DLL: MoneyOutCurr2 (iCurrencyID: Integer; rAmount: Double): Integer

Syntax OLE: MoneyOutCurr2 (iCurrencyID: Integer; rAmount: Double): Integer

Remark: This command used to withdraw cash to the cash drawer. It is the same function as **MoneyOutCurr**, only additional information will be printed on Z report. Only local currency can be withdrawn with this command i.e. variable iCurrencyID should be equal 0.

### OpenCashDrawer

Available in state: IDLE, ZIDLE.

Syntax DLL: OpenCashDrawer (): Integer

Syntax OLE: OpenCashDrawer (): Integer

Remark: This command used to open cash drawer for some reason: to check money amount, exchange banknote etc. This command can't be used to withdraw or enclose money.

### ReceiptSuspend

Available in state: IDLE, ZIDLE.

Syntax DLL: ReceiptSuspend(rAmount: Double): Integer

Syntax OLE: ReceiptSuspend(rAmount: Double): Integer

Remark: This command informs FMB about suspended receipt. When receipt is suspended, the FMB will increase suspended receipt counter and total suspended amount. Those counters aren't cleared after printing of Z report.

### ReceiptRecall

Available in state: IDLE, ZIDLE.

Syntax DLL: ReceiptRecall(rAmount: Double): Integer

Syntax OLE: ReceiptRecall(rAmount: Double): Integer

Remark: This command informs FMB about recall of previously suspended receipt. When receipt is recalled, the FMB will decrease suspended receipt counter and total suspended amount. Those counters aren't cleared after printing of Z report.

## Report function's list

### PrintVatTable

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintVatTable (): Integer

Syntax OLE: PrintVatTable (): Integer

Remark: Prints the VAT table of the FMB.
This isn't a fiscal report. This function only prints current values of VAT table. It can be used to make sure that VAT table is set correctly or to obtain to visual examination current values. VAT table is also printed on Z and X reports.

### PrintCurrencyTable

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintCurrencyTable (): Integer

Syntax OLE: PrintCurrencyTable (): Integer

Remark: Prints the currency table of the FMB.
This isn't a fiscal report. This function only prints current values of currency table. It can be used to make sure that currency table is set correctly or to obtain to visual examination current values. Currency table is also printed on Z and X reports.

### PrintZReport

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintZReport (): Integer

Syntax OLE: PrintZReport (): Integer

Remark: This is keystone of the FMB's fiscal reports. Otherwise it is called as end-of-day report. Usually this command performed on the end of day and writes day's turnovers data to the FMB memory. The FMB state will be switched to **ZIDLE**.

### FinalizeZReport

Available in state: IDLE, ZIDLE.

Syntax DLL: FinalizeZReport (): Integer

Syntax OLE: FinalizeZReport (): Integer

Remark: This command is automatically executed within **PrintZReport** function. It compares CRC32 values of local and FMB's e-journals and if they are equal, erases FMB's e-journal. If for any reason

during execution of **PrintZReport** command Z receipt was successfully printed, but returned an error code (usually error 161 - timeout) then by using this command **PrintZReport** is finished as required.

### PrintXReport

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintXReport (): Integer

Syntax OLE: PrintXReport (): Integer

Remark: Prints current day turnover (X report). This report is similar to **PrintZReport** except that day's turnover data will not be written to the FMB memory. Usually this command performed to check current turnover, cash amount in drawer.

### PrintMiniXReport

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintMiniXReport (): Integer

Syntax OLE: PrintMiniXReport (): Integer

Remark: Prints short X report.
This function is familiar to **PrintXReport** except that day's turnover isn't detailed. Usually this command performed with reason to save receipt paper.

### PrintSumPeriodicReport

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintSumPeriodicReport (pstrDateFrom,pstrDateUntil: PChar): Integer

Syntax OLE: PrintSumPeriodicReport (wstrDateFrom,wstrDateUntil: WideString): Integer

Remark: Prints summarized periodical report.
Variables pstrDateFrom and pstrDateUntil defines interval of the report. The dates must passed in format "YYYY.MM.DD" where YYYY - year, MM -month, DD - day. If month or day is less than 10 then fill spaces with "0" character**.**
Usually performed there are printed the three reports per moth:

1) From 1st to 15$^{th}$
2) From 15$^{th}$ to the last day of month
3) From 1$^{st}$ to the last day of month

### PrintPeriodicReport

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintPeriodicReport (pstrDateFrom,pstrDateUntil: PChar): Integer

Syntax OLE: PrintPeriodicReport (wstrDateFrom,wstrDateUntil: WideString): Integer

Remark: Prints detailed periodical report. In fact this report is combined from copies of Z reports.
Variables pstrDateFrom and pstrDateUntil defines interval of the report. The dates must passed in format “YYYY.MM.DD” where YYYY - year, MM -month, DD - day. If month or day is less than 10 then fill spaces with “0” character.

### PrintSumPeriodicReportByNumber

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintSumPeriodicReportByNumber(iNumberFrom,
iNumberUntil: Integer): Integer

Syntax OLE: PrintSumPeriodicReportByNumber (iNumberFrom,
iNumberUntil: Integer): Integer

Remark: Prints summarized periodical report. This command is similar to **PrintSumPeriodicReport** except instead range of dates, range of Z reports is used.
Variables iNumberFrom and iNumberUntil indicates the range of report and point required numbers of Z reports.

### PrintPeriodicReportByNumber

Available in state: IDLE, ZIDLE.

Syntax DLL: PrintPeriodicReportByNumber (iNumberFrom,
iNumberUntil: Integer): Integer

Syntax OLE: PrintPeriodicReportByNumber (iNumberFrom,
iNumberUntil: Integer): Integer

Remark: Prints detailed periodical report. This command is similar to **PrintPeriodicReport** except instead of range of dates, range of Z reports is used.
Variables iNumberFrom and iNumberUntil indicates the range of report and point required numbers of Z reports.

## Customer display function's list

### CustomerDisplay2

Available in state: ANY.

Syntax DLL: CustomerDisplay2 (pstrLine1, pstrLine2: PChar): Integer;

Syntax OLE: CustomerDisplay2 (wstrLine1, wstrLine2: WideString): Integer;

Remark: Indicates information to the customer display.
This function is similar to Error! Reference source not found.. The MBSetup program defines display type that will be used by the FMB. Variables pstrLine1 and pstrLine2 contains accordingly first and second line of the customer display. If there is used digital display then only first line will be used. If line starts with character or contain character in its body then will be show only digits until than character.

### CustomerDisplayPro

Available in state: ANY.

Syntax DLL: CustomerDisplayPro (pstrCommand: PChar): Integer

Syntax OLE: CustomerDisplayPro (wstrCommand: WideString): Integer

Remark: Indicates information to the customer display.
If there is need to use unlisted customer display or use specific function of display then customer display can be handled through this function.
Variable pstrCommand contains command or data that are redirected to the customer display.

## Non-Fiscal function's list

### BeginNonFiscalReceipt

Available in state: IDLE, ZIDLE.

Syntax DLL: BeginNonFiscalReceipt (): Integer

Syntax OLE: BeginNonFiscalReceipt (): Integer

Remark: Starts non-fiscal receipt and switch the FMB state to **NONFIS**. In this state any text can be printed with **PrintNonFiscalLine** command.

### PrintTare

Available in state: NONFIS.

Syntax DLL: PrintTare (pstrDescription: PChar;
rQuantity,rUnitPrice: Double): Integer

Syntax OLE: PrintTare (wstrDescription: WideString;
rQuantity, rUnitPrice: Double): Integer

Remark: This command is used to print tare on receipt.
Variable pstrDescription contains description of tare. It can exceed 29 characters.
Variable rQuantity contains number of returned quantity of tare.
Variable rUnitPrice contains price of single tare unit.

### PrintTareItemVoid

Available in state: NONFIS.

Syntax DLL: PrintTareItemVoid (pstrDescription: PChar;
rQuantity, rUnitPrice: Double): Integer

Syntax OLE: PrintTareItemVoid (wstrDescription: WideString;
rQuantity, rUnitPrice: Double): Integer

Remark: This command is used to void printed tare on receipt.
Variable pstrDescription contains description of tare. It can exceed 29 characters.
Variable rQuantity contains number of voided quantity of tare. Total amount to void can't exceed current total of tare.
Variable rUnitPrice contains price of single tare unit.

### PrintDepositReceive

Available in state: NONFIS.

Syntax DLL: PrintDepositReceive (pstrDescription: PChar;
rQuantity, rUnitPrice: Double): Integer

Syntax OLE: PrintDepositReceive (wstrDescription: WideString;

rQuantity, rUnitPrice: Double): Integer

Remark: This command is used to print received deposit on receipt.
Variable pstrDescription contains description of deposit. It can exceed 29 characters.
Variable rQuantity contains number of quantity of deposit.
Variable rUnitPrice contains price of single deposit unit.

### PrintDepositRefund

Available in state: NONFIS.

Syntax DLL: PrintDepositRefund(pstrDescription: PChar;
rQuantity, rUnitPrice: Double): Integer

Syntax OLE: PrintDepositRefund(wstrDescription: WideString;
rQuantity, rUnitPrice: Double): Integer

Remark: This command is used to print refunded deposit on receipt.
Variable pstrDescription contains description of deposit. It can exceed 29 characters.
Variable rQuantity contains number of quantity of deposit.
Variable rUnitPrice contains price of single deposit unit.

### PrintBarCode

Available in state: FIS, NONFIS.

Syntax DLL: PrintBarCode (iSystem, iHeight: Integer;pstrBarCode: PChar): Integer

Syntax OLE: PrintBarCode (iSystem, iHeight: Integer;wstrBarCode: WideString): Integer

Remark: Some printers' supports printing of barcodes. This command allows printing a barcode on receipt. There are list of supported barcodes[6].

[6] Barcode support depends on type of connected printer. Printer may not support all listed types of barcodes. In this case will be printed barcode string – not barcode.

| System | Name | Length | Allowed codes (decimal) |
|---|---|---|---|
| 0 | UPC-A | 11≤k≤12 | 48≤d≤57 |
| 1 | UPC-E | 11≤k≤12 | 48≤d≤57 |
| 2 | JAN13 (EAN13) | 12≤k≤13 | 48≤d≤57 |
| 3 | JAN8 (EAN8) | 7≤k≤8 | 48≤d≤57 |
| 4 | CODE39 | 1≤k | 48≤d≤57, 65≤d≤90, 32, 36, 37, 43, 45, 46, 47 |
| 5 | ITF | 1<k (even) | 48≤d≤57 |
| 6 | CODABAR | 1≤k | 48≤d≤57, 65≤d≤68, 36, 43, 45, 46, 47, 58 |

Barcode length can't exceed 235 characters.
Value of variable iSystem corresponds to value in system column from table above. Variable pstrBarCode contains barcode code and must contain only valid characters.
Variable iHeight can contain value from 1 to 255. It has effects on height of printed barcode.

### PrintNonFiscalLine

Available in state: NONFIS.

Syntax DLL: PrintNonFiscalLine (pstrLine: PChar; iAttrib: Integer): Integer

Syntax OLE: PrintNonFiscalLine (wstrLine: WideString; iAttrib: Integer): Integer

Remark: Prints non-fiscal line on receipt.
Variable pstrLine contains text to be printed.
Variable iAttrib defines text output style. **Attributes for lines should be used with caution, because of them line can exceed paper width and will be automatically wrapped**. Variable must define font size and/or additional properties. The font ID and properties are combined by binary OR function. Available codes are showed in the table below.

| Value (HEX) | Description |
|---|---|
| 0x40<br>0x41 | Font size 9x9<br>Font size 7x9 |
| 0x44 | Underline |
| 0x48 | Bold |
| 0x50 | Double height |
| 0x60 | Double width |

### EndNonFiscalReceipt

Available in state: NONFIS.

Syntax DLL: EndNonFiscalReceipt (): Integer

Syntax OLE: EndNonFiscalReceipt (): Integer

Remark: This command finishes non-fiscal receipt started with **BeginNonFiscalReceipt** command. If there is used command for tare sale or deposit then additional lines with change will be printed. The FMB state will be switched to **IDLE**.

## Fiscal function's list

### BeginFiscalReceipt

Available in state: IDLE, ZIDLE.

Syntax DLL: BeginFiscalReceipt (): Integer

Syntax OLE: BeginFiscalReceipt (): Integer

Remark: Starts fiscal receipt and switch the FMB state to **FIS**.

### PrintRecItem

Available in state: FIS.

Syntax DLL: PrintRecItem (pstrDescription: PChar;
rQuantity, rUnitPrice: Double;
iVATID: Integer; pstrDimension: PChar): Integer

Syntax OLE: PrintRecItem (wstrDescription: WideString;
rQuantity, rUnitPrice: Double;
iVATID: Integer; wstrDimension: WideString): Integer

Remark: Print an item on receipt.
Variables contain these values:

- PstrDescription – Item description. Can't exceed 29 characters in length,
- RQuantity – amount of selling item,
- RUnitPrice – price of item unit. Can't be equal 0,
- IVATID – VAT number to corresponding value in the FMB VAT table,
- PstrDimension – item measure unit. Can't exceed 4 characters in length.

### ItemReturn

Available in state: FIS.

Syntax DLL: ItemReturn (pstrDescription: PChar;
rQuantity, rUnitPrice: Double;
iVATID: Integer; pstrDimension: PChar;
rDAInCurrencyPercent, rDAInCurrencyAbsolute: Double):
Integer

Syntax OLE: ItemReturn (wstrDescription: WideString;
rQuantity, rUnitPrice: Double;
iVATID: Integer; wstrDimension: WideString;
rDAInCurrencyPercent,rDAInCurrencyAbsolute: Double):
Integer

Remark: Function used to recall an item of current receipt until receipt isn't finished. This function is equal to **PrintRecItem**, but in addition has two additional variables:

- pstrDescription – Item description. Can't exceed 29 characters in length,
- rQuantity – amount of selling item,
- rUnitPrice – price of item unit. Can't be equal 0,
- iVATID – VAT number to corresponding value in the FMB VAT table,
- pstrDimension – item measure unit. Can't exceed 4 characters in length,
- rDAInCurrencyPercent – indicates amount of percent discount/addition applied for this item. Negative amount correspond to discount and positive to addition. This isn't discount/addition rate in percents – this is equivalent in currency ,
- rDAInCurrencyAbsolute - indicates amount of absolute discount/addition applied for this item.

Note: If item had applied discount/addition, then item price isn't the same as in sale command. The item price must be recalculated to represent applied discount/addition. This is done in three steps:
1 – Calculate item price without discount/addition
(0.8) $ItemTotal = Round(ItemPrice \times ItemQuantity)$
2 – Calculate amount of applied discount/addition. These values also will be used as rDAInCurrencyPercent and rDAInCurrencyAbsolute
(0.9)
a) $Amount_{PercentDiscount/Surcharge} = Round(ItemTotal \times \frac{Percent_{RateDiscount/Surcharge}}{100})$
b)
$Amount_{AbsoluteDiscount/Surcharge} = Round(Amount_{AbsoluteDiscount/Surcharge})$ 3 – Calculate item price with discount/addition. Item price should not be rounded.
(0.10)
$ItemPrice_{WithDiscount/Surcharge} = \frac{ItemTotal + Amount_{Discount/Surcharge}}{ItemQuantity}$

## PrintBarCode

Available in state: FIS, NONFIS.

Syntax DLL: PrintBarCode (iSystem, iHeight: Integer;pstrBarCode: PChar): Integer

Syntax OLE: PrintBarCode (iSystem, iHeight: Integer;wstrBarCode: WideString): Integer

Remark: Some printers' supports printing of barcodes. This command allows printing a barcode on receipt. There are list of supported barcodes[7].

| System | Name | Length | Allowed codes (decimal) |
|---|---|---|---|
| 0 | UPC-A | 11≤k≤12 | 48≤d≤57 |
| 1 | UPC-E | 11≤k≤12 | 48≤d≤57 |
| 2 | JAN13 (EAN13) | 12≤k≤13 | 48≤d≤57 |
| 3 | JAN8 (EAN8) | 7≤k≤8 | 48≤d≤57 |
| 4 | CODE39 | 1≤k | 48≤d≤57, 65≤d≤90, 32, 36, 37, 43, 45, 46, 47 |
| 5 | ITF | 1<k (even) | 48≤d≤57 |
| 6 | CODABAR | 1≤k | 48≤d≤57, 65≤d≤68, 36, 43, 45, 46, 47, 58 |

Barcode length can't exceed 235 characters.
Value of variable iSystem corresponds to value in system column from table above. Variable pstrBarCode contains barcode code and must contain only valid characters.
Variable iHeight can contain value from 1 to 255. It has effects on height of printed barcode.

### PrintCommentLine

Available in state: FIS.

Syntax DLL: PrintCommentLine (pstrLine: PChar; iAttrib: Integer): Integer

Syntax OLE: PrintCommentLine (wstrLine: WideString; iAttrib: Integer): Integer

Remark: Prints comment line on receipt.
Variable pstrLine contains comment to be printed.
Variable iAttrib defines text output style. Variable must define font size and/or additional properties. **Attributes for lines should be used with caution, because of them line can exceed paper width and will be automatically wrapped**. The font ID and properties are combined by binary OR function. Available codes are showed in the table below.

| Value (HEX) | Description |
|---|---|
| 0x40 | Font size 9x9 |

[7] Barcode support depends on type of connected printer. Printer may not support all listed types of barcodes. In this case will be printed barcode string – not barcode.

| | |
|---|---|
| 0x41 | Font size 7x9 |
| 0x44 | Underline |
| 0x48 | Bold |
| 0x50 | Double height |
| 0x60 | Double width |

### DiscountAdditionForItem

Available in state: FIS.

Syntax DLL: DiscountAdditionsForItem (iDAType: Integer;rAmount: Double): Integer

Syntax OLE: DiscountAdditionsForItem (iDAType: Integer;rAmount: Double): Integer

Remark: Applies discount/addition for the last item send to the FMB by **PrintRecItem** function.
Variable iDAType can contain one of these values:

| Value (DEC) | Description |
|---|---|
| 1 | Percent |
| 2 | Absolute |

Variable rAmount contain percent or sum value of discount/addition. Negative value corresponds to discount, positive to addition.

### DiscountAdditionForReceipt

Available in state: FIS.

Syntax DLL: DiscountAdditionsForReceipt (iDAType: Integer;rAmount: Double): Integer

Syntax OLE: DiscountAdditionsForReceipt (iDAType: Integer;rAmount: Double): Integer

Remark: Applies discount/addition for the current total of receipt. This function can be issued after first sold item, and after this function item sale also is available.
Variable iDAType can contain one of these values:

| Value (DEC) | Description |
|---|---|
| 1 | Percent |
| 2 | Absolute |

Variable rAmount contain percent or sum value of discount/addition. Negative value corresponds to discount, positive to addition.

### EndFiscalReceiptCurr

Available in state: FIS.

Syntax DLL: EndFiscalReceiptCurr (rCash,

rCredit1, rCredit2, rCredit3, rCredit4,
rCurrency1, rCurrency2, rCurrency3: Double):Integer;

Syntax OLE: EndFiscalReceiptCurr (rCash,
rCredit1, rCredit2, rCredit3, rCredit4,
rCurrency1, rCurrency2, rCurrency3: Double):Integer;

Remark: Ends fiscal receipt ant switch the FMB state to IDLE.
Variables rCash, rCredit1, rCredit2, rCredit3 and rCredit4 contain payment amounts.

- rCash – cash payment amount;
- rCredit(n) –non-cash payment amount;
- rCurrency(n)-cash payment amount.

This command allows performing receipt payment in currency.
There are four currency types supported by the FMB. But one type is permanently set to national currency. For more information refer to **GetFiscalInfo** command. Only set currency types can be used in payment, otherwise amount of currency must be sent as zero.
National currency is sent in rCash variable.
Can be used any combinations of payments, but total sum of non-cash amount must be equal or less than total of receipt. If sum of non-cash payments are less than total of receipt then lacking sum will be deducted from cash payment amount.
Change always will be returned in national currency despite amount of foreign currencies.

### GoodsReturnCurr

Available in state: FIS.

Syntax DLL: GoodsReturnCurr (rCash,
rCredit1, rCredit2, rCredit3, rCredit4,
rCurrency1, rCurrency2, rCurrency3: Double):Integer;

Syntax OLE: GoodsReturnCurr (rCash,
rCredit1, rCredit2, rCredit3, rCredit4,
rCurrency1, rCurrency2, rCurrency3: Double):Integer;

Remark: Ends fiscal receipt as items return receipt ant switch the FMB state to IDLE.
Variables rCash, rCredit1, rCredit2, rCredit3 and rCredit4 contain payment amounts.

- rCash – cash payment amount;
- rCredit(n) –non-cash payment amount.
- rCurrency(n)-cash payment amount.

This function used is when customers want to return an item.
Can be used any combinations of payments, but total sum of them

must be equal to total of receipt.
Used payment types must be the same as on the item sale receipt.
Also the FMB cash account (cash in drawer) must contain required sum.

## E-Journal function's list

### PrintCopyOfLastReceipt

Available in state: ZIDLE, IDLE.

Syntax DLL: PrintCopyOfLastReceipt (): Integer

Syntax OLE: PrintCopyOfLastReceipt (): Integer

Remark: Prints copy of the last receipt in the non-fiscal mode.

### PrintCopyOfReceipt

Available in state: ZIDLE, IDLE.

Syntax DLL: PrintCopyOfReceipt (iReceiptFrom, iReceiptUntil: Integer): Integer;

Syntax OLE: PrintCopyOfReceipt (iReceiptFrom, iReceiptUntil: Integer): Integer;

Remark: Prints copy of the receipt/receipts in the non-fiscal mode.
Variables iReceiptFrom and iReceiptUntil contains numbers of printed receipts. Those numbers should be from currently available range of receipt stored in current e-journal. Range of currently available receipts can be determined from values received from **GetFiscalInfo** function, parameter 900.

### GetCopyOfReceipt

Available in state: ZIDLE, IDLE.

Syntax DLL: GetCopyOfReceipt (iReceiptFrom, iReceiptUntil: Integer;
var pstrResult:PChar): Integer

Syntax OLE: GetCopyOfReceipt (iReceiptFrom, iReceiptUntil: Integer;
var wstrResult: WideString): Integer;

Remark: Retrieves receipt/receipts from FMB's e-journal.
Variables iReceiptFrom and iReceiptUntil contains numbers of printed receipts. Those numbers should be from currently available range of receipt stored in current e-journal.

### FormatCard

Available in state: ZIDLE, IDLE.

Syntax DLL: FormatCard (): Integer

Syntax OLE: FormatCard (): Integer

Remark: Formats internal FMB memory card. This card is used as permanent storage for e-journal backup. Before formatting card, currently stored files must be moved to another storage media.

### PrintCardDirector

Available in state: ZIDLE, IDLE.

Syntax DLL: PrintCardDirectory (): Integer

Syntax OLE: PrintCardDirectory (): Integer

Remark: Prints list of current e-journal files on internal FMB memory card.

# Appendix A

Result code list

There is the full list of possible FMB and EmpiFis errors. Errors can be divided in such categories:

- Serial. Codes from 4 to 7,
- Command. Codes from 16 to 60,
- Software. Codes above 160

Serial level error indicates communication or low-level command check error from FMB. They can be returned by all commands.

Command level error indicates specific command condition.

Software level errors are produced by EmpiFis module. They indicate of a preparation error before sending a command to the FMB.

| Result code | Value | | Description |
|---|---|---|---|
| | DEC | HEX | |
| FMB_OK | 0 | 0x00 | Success, no error |
| ERR_LENGTH | 4 | 0x04 | Invalid length of the FMB package. The package received but received length doesn't match to value indicated in the package. |
| ERR_DATA | 5 | 0x05 | Invalid data in the FMB package. The package contains code that isn't allowed e.g. non-ASCII or control codes, |
| ERR_XOR | 6 | 0x06 | Invalid checksum of the FMB package. |
| ERR_ETX | 7 | 0x07 | Didn't receive symbol of end of the FMB package. |
| ERR_ILLEGAL | 16 | 0x10 | Unknown or unsupported command |
| ERR_IDLE_STATE | 17 | 0x11 | Function can't be executed in the FMB IDLE state |
| ERR_NONFIS_STATE | 18 | 0x12 | Function can't be executed in the FMB NONFIS state |
| ERR_FIS_STATE | 19 | 0x13 | Function can't be executed in the FMB FIS state |
| ERR_HARDWARE_STATE | 20 | 0x14 | Hardware error. There are problems with the FMB peripherals: printer or customer display. |
| ERR_PARAMETERS | 21 | 0x15 | Invalid parameters in the FMB package. Not enough parameters for command or types of parameters contain invalid values. |
| ERR_ITEM_DESC_LENGTH | 22 | 0x16 | Item description to long. The description can't exceed 39 characters |
| ERR_ITEM_OUANTITY | 23 | 0x17 | Invalid quantity value. Value can't be negative |
| ERR_ITEM_PRICE | 24 | 0x18 | Invalid price value. Value can't be negative |

| | | | |
|---|---|---|---|
| ERR_VAT | 25 | 0x19 | Invalid VAT number. Supported numbers from 0 to 3. The VAT number 3 value isn't changeable and equals to zero. |
| ERR_ITEM_DIM | 26 | 0x1A | Invalid item measure unit value. The unit name can't exceed 4 characters |
| ERR_DEFICIENT_PAYMENT | 27 | 0x1B | Not enough payment sums for receipt. |
| ERR_OVERPAYMENT_CREDIT | 28 | 0x1C | Payment by credit sums exceed receipt total sum. |
| ERR_ITEM_DISCOUNT | 29 | 0x1D | Invalid discount/addition for item. Possible clauses: percent discount exceed 100% value, fixed discount exceed item price and etc. |
| ERR_DISCOUNT_TYPE | 30 | 0x1E | Invalid discount/addition type. |
| ERR_COMMENT_LENGTH | 31 | 0x1F | Comment line too long. |
| ERR_PRINTER | 32 | 0x20 | Printer error |
| ERR_DISPLAY | 33 | 0x21 | Customer display error |
| ERR_PRICE_AMOUNT_OVERFLOW | 34 | 0x22 | Overflow of product of item price and quantity. This product can't exceed 99999.99 |
| ERR_FLASH_WRITE | 35 | 0x23 | Can't write to the FMB memory |
| ERR_NOT_FISCAL | 36 | 0x24 | The FMB isn't fiscalized. Can't access some of the FMB reports |
| ERR_BAD_DATE | 37 | 0x25 | Invalid date format |
| ERR_REPORT_NOT_FOUND | 38 | 0x26 | Can't find report. There isn't report for required interval of dates. |
| ERR_FLASH_ERASE | 39 | 0x27 | Can't erase or prepare to write the FMB memory |
| ERR_DISCOUNT_RECEIPT | 40 | 0x28 | Invalid discount/addition for receipt. Possible clauses: percent discount exceed 100% value, fixed discount exceed receipt total and etc. |
| ERR_NO_ITEMS | 41 | 0x29 | No items in current receipt. Can't apply discount/addition. |
| ERR_CANT_RETURN | 42 | 0x2A | Can't decline item. There are no items in current receipt. |
| ERR_OVER_ADDITION_PERCENT | 43 | 0x2B | Percent addition to big |
| ERR_OVER_DISCOUNT_PERCENT | 44 | 0x2C | Percent discount to big |
| ERR_OVER_ADDITION_FIXED | 45 | 0x2D | Absolute addition to big |
| ERR_OVER_DISCOUNT_FIXED | 46 | 0x2E | Absolute discount to big |
| ERR_NO_MONEY_FOR_DA | 47 | 0x2F | Not enough cash in drawer |
| ERR_ZERO_TOTAL | 48 | 0x30 | Current receipt total is equal zero |
| ERR_PAYMENT_NOT_EQUAL | 49 | 0x31 | Return receipt payment sum not equal receipt total sum |
| ERR_DEFICIENT_CASH_DRAWER | 50 | 0x32 | Not enough cash in drawer |
| ERR_UNIQUE_FMB_NUMBER | 51 | 0x33 | Invalid number of the FMB. Can't allow GoodsReturn function or disables this function |
| ERR_NOTALLOW_GOODS_RETURN | 52 | 0x34 | GoodReturn function isn't allowed. |
| | | | |
| ERR_ALREADYFISCAL | 54 | 0x36 | The FMB already fiscalized. |

| | | | |
|---|---|---|---|
| ERR_NOT_SLIP_PRINTER | 55 | 0x37 | Current printer hasn't SLIP. Tried to start fiscal or non-fiscal receipt to SLIP. |
| ERR_RECEIPT_AMOUNT_OVERFLOW | 56 | 0x38 | Current receipt total exceeded. Receipt tota sum can't exceed 9900000 |
| ERR_FLASH_FULL | 57 | 0x39 | The capacity of FMB memory is near end of already full |
| ERR_YEAR_VALUE | 58 | 0x3A | Invalid year value in date |
| ERR_MONTH_VALUE | 59 | 0x3B | Invalid month value in date |
| ERR_DAY_VALUE | 60 | 0x3C | Invalid day value in date |
| | | | |
| ERR_OUT | 160 | 0xA0 | Can't send package to the FMB. No connection to the FMB. |
| ERR_RECEIVE | 161 | 0xA1 | Error occurred while received package from the FMB. No answers from the FMB, received package integrity problem. |
| ERR_NEGATIVE_CASH | 162 | 0xA2 | Negative cash amount. Tried to withdraw or deposit to cash drawer negative amount of cash. |
| ERR_BARCODE_LENGTH | 163 | 0xA3 | Invalid barcode length. Barcode's code too long or too short for selected barcode system. |
| ERR_BARCODE_SYSTEM | 164 | 0xA4 | Unsupported barcode system |
| ERR_BARCODE_CHAR | 165 | 0xA5 | Unsupported characters in code of barcode |
| ERR_BARCODE_HEIGHT | 166 | 0xA6 | Barcode height is out of supported range |
| ERR_LOGO_NOT_FOUND | 201 | 0xC9 | Graphical logo file not found. Ordered to upload file didn't exists |
| ERR_LOGO_INVALID_FILE_LENGTH | 202 | 0xCA | Invalid format of graphical logo file. Requirements for this file: monochrome BMP file with graphics size 160x40 pixels |
| ERR_JRN_CONVERTNUMEBRCRC | 300 | 0x12C | Can't convert string values of e-journal's last index and CRC32 to numeric |
| ERR_JRN_ADDTOJOURNAL | 301 | 0x12D | Can't add receipt to local e-journal file. CRC32s of local file and FMB are not equal, or can't write to local e-journal file |
| ERR_JRN_COUNTCRCHASH | 302 | 0x12E | Invalid hash of local e-journal index and CRC32. Manually edited e-journal subsidiary file |
| ERR_JRN_JOURNALHASH | 303 | 0x12F | Invalid hash of local e-journal file. Manually edited e-journal file |
| ERR_JRN_FMBCOUNT | 304 | 0x130 | Local and FMB e-journal indexes of last receipt are different |
| ERR_JRN_FMBCRC32 | 305 | 0x131 | Local and FMB ejournal CRC32 is different |
| ERR_JRN_BACKUPJOURNAL | 306 | 0x132 | Can't move local e-journal to archive |
| ERR_JRN_STATEINVALID | 307 | 0x133 | Invalid state of FMB. |
| ERR_JRN_UNAVAILABLE_ARHCIVE | 308 | 0x134 | Archive not found. Path to archive is invalid |
| ERR_JRN_FREESPACE_ARHCIVE | 309 | 0x135 | Not enough free space for archive (minimal free space required is 4Mb) |