The API description of the GpeEcrCommLib.dll

Version 1.04

Created	1.7.2015
Last change	21.3.2016
Author	Pavel Perman

Pos Protocol ver. 12.18, B0

Release history

Date	Version	Description
1.7.2015	1.00	Draft – base functionalities
31.7.2015	1.01	Revision related to the reference implementation
3.8.2015	1.02	Revision related to the first integration
15.1.2016	1.03	Revision hypertext links
21.3.2016	1.04	Revision of return codes

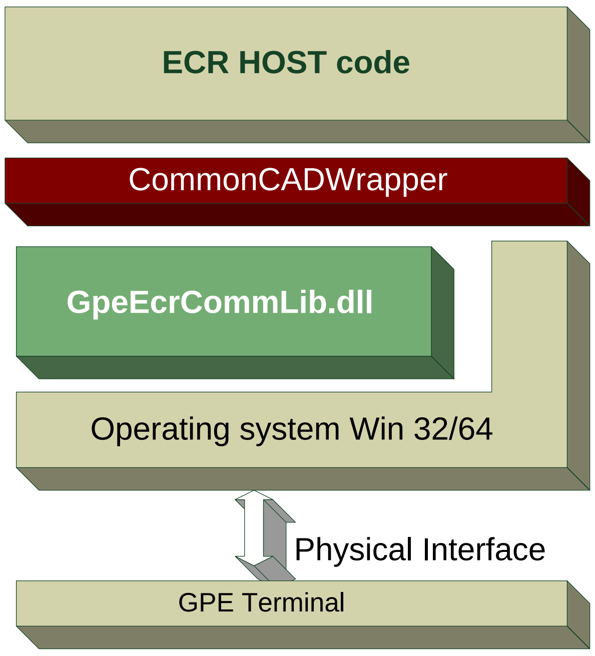
Content

Integration schema

This part shows the mechanism of integration the library into a host code. The easiest way of integration is using the predefined the source code but it is not necessary. The source can be used as an example of the integration. If the CommonCADWrapper module is not supplied it is better to do it on customer site. It is possible to access direct to GpeEcrCommLib API of course.

The open source code (Module) named CommonCADWrapper has several roles:

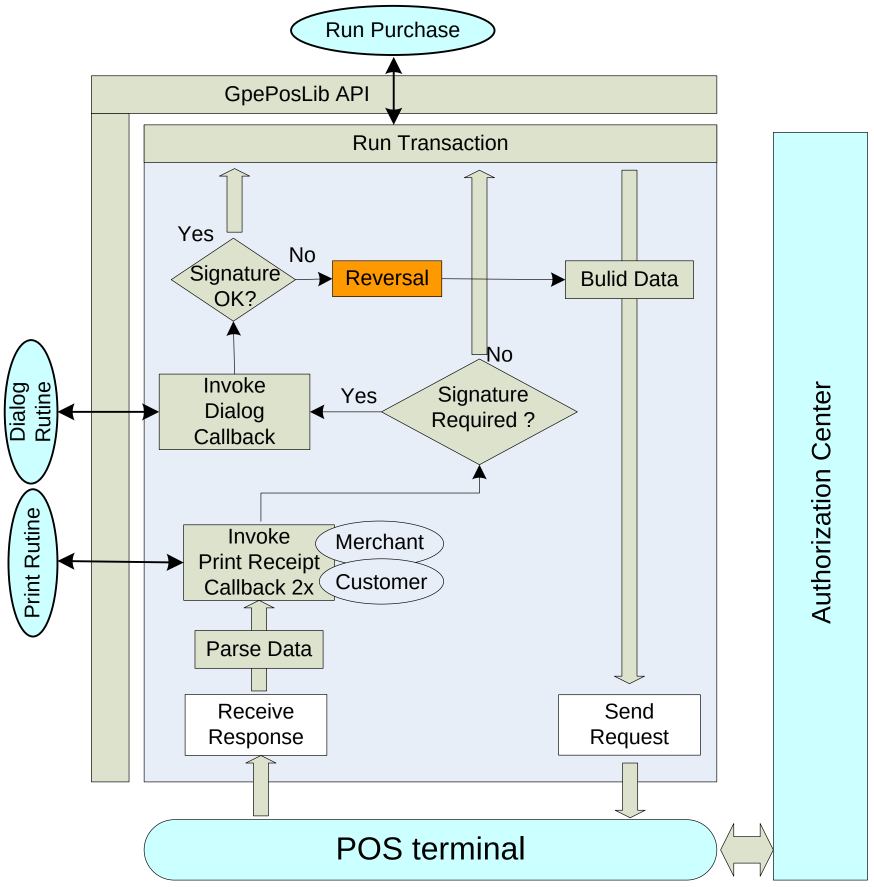
- To shield library specific interface and define common interface for the client code
- Facilitate parameters manipulation and 100% backwards compatibility
- Creating independent intermediate layer for other logic and specific functions aimed to this purposes (checking and conversion data, etc).



Img 1. Integration schema

Processing schema

This part of the specification describes the specific transaction flow. The payment is the base flow which it has to be implemented and it is the most difficult than other transactions. The points of the flow are described below.



Img 2. Processing schema

- Start transaction by call API function RunTransaction. The library build the POS request and sends it to the POS.
- The POS terminal processes the transaction and sends back the response.
- The library parse the data and check if the print data is available. If yes, the library invokes the callback for printing data.
- The library checks if the check signature is required or not. If yes, the library invokes the signature dialog.
- If the dialog is confirmed the library ends the transaction. If the dialog is not confirmed the library runs the reversal transaction internally and the processing flow is the same as previous. Library check the response data and probably invoke the print callback and ends the transaction (signature is no required for the reversal).

Transaction interface

This interface encapsulates the group of transactions which are passed to the terminal through the specific communication protocol (designed by GPE). Every transaction is identified by its class ID. Each transaction can be executed by call function *PosLib_RunTransaction*.

Restrictions

The library interface is created by the several functions. These functions are described in the several chapters below. The description of these function doesn't specify the calling conversion. The calling convention is __stdcall the base but for the more portability it is possible to require compilation with another calling convention like __cdecl or other. The __stdcall is the base due to the library exports the callback functions. Several host technologies doesn't support another calling convention for callbacks, e.g. C#.

The Library is not designed for multi-thread processing. The library use the hardware devices which is not possible to use simultaneously. The library can run inside only one process or thread.

Transaction class table

Class ID	***Transaction***	***Class description***
1	Purchase	Normal purchase or purchase
2	Purchase with Cashback	As the purchase plus cashback amount
3	Refund	Normal Refund
4	Void	Reversal of the last transaction – **not supported yet**
5	Pre-Authorization	Amount Preauthorization – **not supported yet**
6	Authorization Completion	Completion of the previous preauthorization – **not supported yet**
7	CloseTotals	It summarizes the transactions and it clears the totals on terminal and the authorization HOST.
8	SubTotals	It summarizes the transaction but it doesn't clear the totals. It's for common checks through day.

int **GpePosLib_RunTransaction** (int classId, int rootId)
The function executes the transaction on the terminal according to the specific class Id. The input parameters are in the specific TLV tree format which it is not binary dependent. The parameters of transaction are in the table below for each class ID. @Parameters: *[In] classId* Identification of the transaction. The types and count of values are defined according the transaction table. *[In] rootId* reference to Input/Output parameter TLV tree. The parameters of the tree are listed in the ref. *Parameter interface.* @Return: *Return codes table*

Transaction response codes

TrxResult	Description
Approved codes [0-10]	
0	Transaction approved
10	Transaction approved for a partial amount
Declined codes [50 - 100]	
50	Transaction declined (On the host, by the processing, timeout, etc.)
51	Unable to connect with Central Systems
53	A response from the Central System was not delivered
55	Decline by chip card without authorization
56	Declined by Authorization system (online transaction)
60	Transaction was cancelled by the operator
61	Transaction was cancelled by the customer
62	Customer rejected transaction due to amount
63	Processing exceeded the designated time limit
70	Transaction was not cancelled
71	Currency not supported (Multicurrency)
100	Transaction not supported or not allowed
101	Invalid card
107	Error in MAC
200	Mandatory field missing
500	Internal POS Application Error

The response codes are sent form the terminal and is not generated by the DLL component.

Mandatory and optional fields according to class ID. The details about the fields are chap.*Tag description*.

Purchase [class 1]			
Input elements		***Output elements***	
Mandatory	***Optional***	***Mandatory***	***Optional***
Amount	Currency code	Terminal ID	Amount
	Invoice Number	TimeStamp	Expiration date
		TrxResult	Card name
		Authorization code	Unique Transaction ID
		Card PAN [Masked]	AID

Purchase with Cashback [class 2]			
Input elements		***Output elements***	
Mandatory	***Optional***	***Mandatory***	***Optional***
Amount	Currency code	Terminal ID	Amount
Cashback Amount	Invoice Number	TimeStamp	Expiration date
		TrxResult	Card name
		Authorization code	Unique Transaction ID
		Card PAN [Masked]	AID

Refund [class 3]			
Input elements		***Output elements***	
Mandatory	***Optional***	***Mandatory***	***Optional***
Amount	Currency code	Terminal ID	Amount
		TimeStamp	Expiration date
		TrxResult	Card name
		Authorization code	Unique Transaction ID
		Card PAN [Masked]	AID

Close totals[class 7]			
Input elements		***Output elements***	
Mandatory	***Optional***	***Mandatory***	***Optional***
	Summary	Terminal ID	Summary
		TimeStamp	
		TrxResult	

Subtotals [class 8]			
Input elements		***Output elements***	
Mandatory	***Optional***	***Mandatory***	***Optional***
	Summary	Terminal ID	Summary
		TimeStamp	
		TrxResult	

Register callback methods

These functions has to be registered by external hosted code. These functions are invoked inside the transaction process. These callback methods ensure the signature checks and printing receipt on the ECR in time when the transaction flow needs. The display callback is designed for transferring terminal screen into the ECR and this callback could not be registered. Registering printing and dialog callbacks are mandatory.

`int` *(* **pFcDispCb**)* (`char *pText`, `enum codepage`, `int flgTst`) - **It's not supported yet**

This data type is used for call function which it is dedicated for display messages. The message are transferred from the terminal to ECR. This functionality is not supported by the POS protocol yet. For this function timeout is not defined. The message could be displayed on the status line/windows form until new message comes.

@Parameters:

[In] pText	pointer to formatted text
[In] codepage	codepage identification ref. *enum CP_IDX_E*
[In] flgTst	Test flag, if this flag is set the function return 0 without any action. The flag is set only in registration process for checking validity of the function reference.

@Return: 0 – (success) Display message Ok

Other – Failed

`int` *(* **pFcPrnCb**)* (`char *pText`, `enum codepage`, `int imd`, `int flgTst`) - **mandatory**

This data type is used for call function which it is dedicated for printing tickets on the ECR. The text is formatted for the specific codepage which is passed in parameter codepage.The codepage is transferred from the POS terminal.

@Parameters:

[In] pText	pointer to formatted text
[In] codepage	codepage identification ref. *enum CP_IDX_E*
[In] imd	immediate printing, the text must be print immediate. It can be a transaction receipt or warning ticket.
[In] flgTst	Test flag, if this flag is set the function return 0 without any action. The flag is set only in registration process for checking validity of the function reference.

@Return: 0 – (success) Printer Ok

Other – Failed

`int` *(* **pFcDlgCb**)* *(*`int dialogId`, `int flgTst`*)* **- mandatory**
This data type is used for call function which it is dedicated for display dialogs according to its dialog ID. The dialog callback function is for getting reaction of the operator on the library events (especially for signature check). The dialog is identified by its ID. It is the best way how to ensure the correct text localization to different languages. All text are stored in the host Ecr program and library invokes only the specific one according to the Id. @Parameters: *[In] dialogId* Dialog id identifies the real dialog which the library needs to decide. *@Return*: 0 – (success) dialog confirmed Other – Failed or dialog not confirmed

Supported dialogs

Dialog ID	Title	Text pattern
1	The signature comparison	Please compare the signature on the receipt and on the card. Are these signatures the same? Yes/No

Supported codepages in table (enum CP_IDS_E)

Idx	Codepage	Description
1	ISO-8859-1	Use for English texts (display, printing tickets, etc.)
2	ISO-8859-2	Used for Czech, Slovak, Polish, Hungarian texts (display, tickets,
3	ISO-8859-3	Used for Turkish texts (display, printing tickets, etc.)
5	ISO-8859-5	Used for Bulgarian texts (display, printing tickets, etc.)

Functions below is used for register callback methods

`int` **GpePosLib_RegisterPrintCB *(*** `pFcPrnCb pPrinterCB`***)***
The function registers the callback for printing the text while the transaction is processed (receipts or warning tickets). @Parameters: *[In]* pPrinterCB pointer to function which prints the text on the ECR printer. *@Return*: *Return codes table*

`int` **GpePosLib_RegisterDisplayCB *(*** `pFcDispCb pDispCB` ***)***
The function registers the callback for display the text while the transaction is processed (The message from the screen of the connected terminal). @Parameters: *[In]* pDispCB pointer to function which displays text on the ECR screen. *@Return*: *Return codes table*

`int` **GpePosLib_RegisterDialogCB *(*** `pFcDlgCb pDialogCB` ***)***
The function registers the callback for display the dialog on the ECR. The dialog is intended to invoke on end of the transaction for check the signature (if the transaction flow requires it) but it can be used for another purpose in a future. @Parameters: *[In]* pDialogCB pointer to function which displays text on the ECR screen. *@Return*: *Return codes table*

Administrative interface

This part of guide contains specific functions which are dedicated for DLL administration (Initialization, load parameters, close and etc).

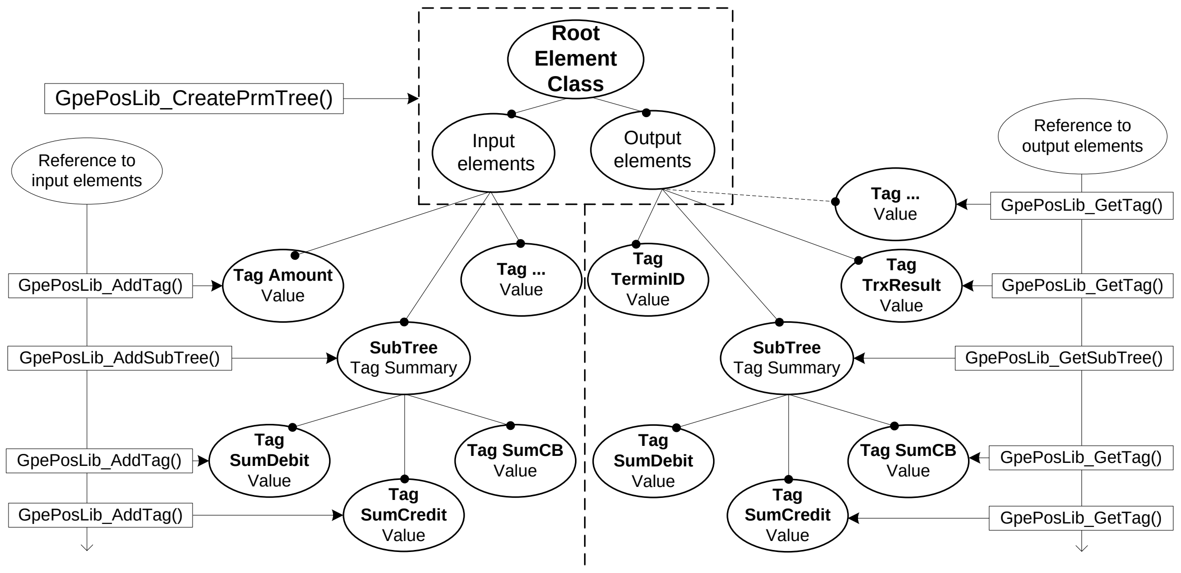
`int` **GpePosLib_OpenLibrary *(*** `char *pPathCfg` ***)***
The function initializes the transaction context and load the parameter file. @Parameters: *[In]* pPathCfg null-terminated string contains the path to the configuration file. If it is not specified the config file is searched in the same folder as DLL. *@Return*: *Return codes table*

`int` **GpePosLib_CloseLibrary *(*** `void` ***)***
The function close all opened peripherals and release allocated memory. It belongs to the good programmer practice to call it. @Parameters: none *@Return*: *Return codes table*

`int` **GpePosLib_GetVersion (**`void`**)**
The function returns the version number of the loaded library. The version of the library has the MS format like: 1.2.3.4 = 0x01020304. @Parameters: none *@Return*: version number in binary format.

Parameter interface

All parameters are passed by TLV tree mechanism. The root is created by the function call GpePosLib_CreatePrmTree. This function returns id of structure which contains the reference to input and output elements subtree. This approach is more flexible that a static structure and easier than a XML. It's assumed that the exported interface could be changed related to new properties, functions or fixing bugs as well. This mechanism allows to modify application parameter interface without any direct impact to backwards compatibility, stability and robustness. The TLV tree is implemented statically due to minimalize problem with memory leaks. This approach doesn't allow to create several trees (by calling the CreatePrmTree func.) simultaneously. It is possible to create only one tree for the current transaction. On the end of the transaction is necessary to call function to release tree which re-initialize the static content of the tree. The parameter application interface is designed for free migration to dynamic allocation of the parameter tree (in a future) regardless to this parameter interface.



Img. 3. Structure of the parameter tree.

`int` **GpePosLib_CreatePrmTree *(*** `int classId` ***)***

The transaction create the parameter tree which is used for the input and output parameters of *GpePosLib_RunTransaction*. The parameter tree MUST be created for each transaction. There is no possible to use the same reference for next transaction. After the transaction is finished it MUST be called *GpePosLib_ReleasePrmTree()* function.

@Parameters:

[In] classId Class id of the prepared transaction.

@Return: >0 - pointer to rootId of the new tlvTree

0 - failed

`int` **GpePosLib_GetInputTree *(*** `int rootId` ***)***

The function retrieves the input part of the parameter tree. This part specifies the input parameters.

@Parameters:

[In] rootId rood id of the tree which is returned by *GpePosLib_CreatedPrmTree()*

@Return: =>0 - id of the input parameter tree.

<0 error, *Return codes table*

`int` **GpePosLib_GetOutputTree *(*** `int rootId` ***)***

The function retrieves the output part of the parameter tree. This part specifies the output parameters. Detail of the transaction.

@Parameters:

[In] rootId rood id of the tree which is returned by *GpePosLib_CreatedPrmTree()*

@Return: =>0 - id of the input parameter tree.

<0 error, *Return codes table*

`int` **GpePosLib_GetIntTag** ***(***`int treeId, int tag`***)***
The function retrieve the int value of the tag which was previously set by *GpePosLib_AddIntTag().* @Parameters: *[In]treeId* the treeId is returned by function call *GpePosLib_GetInputTree () or GpePosLib_GetOutputTree().* *[In]* tag tag number according to the table *Tag description.* @Return: >=0 is expected value of the tag Error: *Return codes table (especially GPE_APP_TAG_...)*

`int` **GpePosLib_AddIntTag** ***(***`int treeId, int tag, int value`***)***
The function add the integer value of the tag. The tag specifies the type of parameter. @Parameters: *[In]treeId* the treeId is returned by function call *GpePosLib_GetInputTree () or GpePosLib_GetOutputTree().* *[In]* tag tag number according to the table *Tag description.* *[In] value* the value of the parameter. *@Return*: 0 - OK *Return codes table (especially GPE_APP_TAG_...)*

`int` **GpePosLib_GetStringTag** ***(***`int treeId, int tag, char *pValue`***)***
The function retrieve the string value of the tag which was previously set by *GpePosLib_AddStringTag().* @Parameters: *[In]treeId* the treeId is returned by function call *GpePosLib_GetInputTree () or GpePosLib_GetOutputTree().* *[In]* tag tag number according to the table *Tag description.* *[In] pValue* value of the tag @Return: 0 - OK *Return codes table (especially GPE_APP_TAG_...)*

`int` **GpePosLib_AddStringTag** ***(***`int treeId, int tag, char *pValue`***)***

The function set the string value into the tag. The tag specifies the type of parameter.

@Parameters:

[In]treeId	the treeId is returned by function call *GpePosLib_GetInputTree () or GpePosLib_GetOutputTree().*
[In] tag	tag number according to the table *Tag description*
[In] value	the value of the parameter.

@Return: 0 - OK

Return codes table (especially GPE_APP_TAG_...)

`int` **GpePosLib_AddSubTree** ***(***`int treeId, int tag`)

The function add the tag to parameter tree which hold the specific parameter value.
@Parameters:

[In] tree	parent Tree id returned by function call *GpePosLib_GetInputTree () or GpePosLib_GetOutputTree()*
[In] tag	tag number according to the table *Tag description.*

@Return: !0 - pointer to created new TLV sub tree

0 – failed

`int` **GpePosLib_GetSubTree** ***(***`int tag`***)***

@Parameters:

[In] tag	tag number according to the table *Tag description.*

@Return: !0 – returns the tree id of the subTree previously created by GpePosLib*_AddSubTree().*

0 – failed or it doesn't exist

`int` **GpePosLib_ReleaseTree** ***(***`int rootId`***)***

The function releases the parameter tree which has been created by previous call function *GpePosLib_CreatePrmTree()*. The function call ensure re-initialize static TLV tree.

@Parameters:

[In] rootId	*root of the tree*

@Return: 0 - OK

Return codes table

Tag description

This part descripts parameters tag which are used for execution of transactions. All tag are intended as Integer or ANSI string. There is no possible to create other type of tags. The format of the data is specified by the tag. The valid data is in the table below. The data with integer type added as a string value causes error and vice versa.

TAG specification

TAG	Name	Type	Len	Description
0	Root	N	1	[I/O] - Default
1	Terminal ID	AN	String 6-8	[O] - Terminal identification. This number is registered in the authorization center.
2	Timestamp	String N	String 12	[O] - Timestamp of the transaction used by the protocol messages.
3	Amount	N	int	[I/O] - Amount must be >0 and < 9 999 999, 99. Amount must be without decimal places. If the currency supports decimal places the amount must be multiplied correspondingly (i.e. for CZK x 100). Max. 9 999 999,99
4	CBAmount	N	int	[I] - CashBack amount. The format is the same as Amount
5	Currency	N	int	[I] - Currency code according to *ISO 4217* (CZK 203, EUR 978, …)
6	Authorization Code	V	String 1-8	[I/O] - The authorization code identifies the transaction on the authorization host.
7	CardName	V	String 1-20	[O] - It specifies the name of card which it was used for the transaction.
8	PAN	V	String 9-19	[O] – Primary Account Number
9	respCode	N	Int	[O] - Transaction response code. Detail is in the Transaction response codes table.
10	InvoiceNumber	N	String 1-10	[I] - The invoice number is place on the receipt ticket and can be propagated to host.
11	ExpirDate	N	String 4	[O] – expiration date of the used card
12	CodePage	N	int	[O] – codepage of the receipt text. The value is index into the *Supported codepages table*.
13	AID	AN	String	[O] - It specifies the chip card application ID

			1-24	
14	SequenceNo	N	String 1-10	[I/O] - It specifies the number for preauthorization and completion.
15	TrxId	N	String 12	[O] - Unique transaction ID generated on the terminal for each approved transaction.
16	Summary	struct	----	[I/O] – On this tag is necessary to create subtree which it holds the group of summery tags. Totals summary which is compared with the host. The description is in the table below. If the Summary tag not used (it's optional) terminal used its own summary.

*** note:** [O] means input parameter, [I] – means output parameter, [I/O] – both.

SubTree tag contains the all the summary tags.

TAG	Name	Type	Len	Description
100	RootSummeryTree	---	---	The root tag
101	DebitCount	N	Int	Count of the Purchase transactions
102	DebitSum	V	String 19	String value contains summary of the purchase trx. It must be signed and left-padded by 0 to full 18 chars.. Example: "*+000000000000200000*" 2000,00 Kč It contains also cashback amount.
103	CreditCount	N	Int	Count of the Refund transactions
104	CreditSum	V	String 19	String value contains summary of the refund trx. It must be signed and left-padded by 0 to full 18 chars..
105	CBCount	N	Int	Count of the Cashback transactions.
106	CBSum	V	String 19	String value contains only cashback summary amount.

Example:

2 debit transaction in the total amount of CZK 2000.00
1 credit transaction for CZK 100.00
1 Cashback for CZK 300.00

```
101 = 2          102 = +000000000000200000
103 = 1          104 = +000000000000010000
105 = 1          106 = +000000000000030000
```

Tag type

Abbr.	Name	Description
A	Alfa	Containing A-Z, a-z characters
N	Numeric	Field with Numerical characters
AN	Alfa-Numeric	0-9, A-Z, a-z
V	Visible	hexadecimal values 0x20 up to 0x7E

Return codes table

Return codes are valid for all functions.

Return code	***Description***
<0 / -49>	***Common errors***
0	Success – no error occurred
-1	No input parameters
-2	Library not found or it cannot be loaded
-3	Mandatory callbacks (Dialog, Print) are not registered
-10	General error – not further specified
<-50/-99>	***Application errors***
-50	Application interface was not initialized
-51	Application data is not available
-52	Application data is not valid
-53	Bad or mismatches input parameters – mandatory data is missing
-70	(Tlv tree) Tag general error
-71	(Tlv tree) Tag format error
-72	(Tlv tree) Tag not found
-73	(Tlv tree) Tag unknown format
<-100 / -199>	***Transaction errors***
-100	Transaction process interrupted
-108	Transaction is in the progress or terminal is busy
<-200 / -299>	***Communication errors***
-200	Terminal not found or terminal not responding
-201	There is not response from the terminal – Response timeout
-202	General communication error with the terminal – Communication corrupted (CRC, Format errors)
-203	Open/Initialization of the communication channel error
<-300 / - 500>	***Unknown, undefined or system error – the reason is not specified, the transaction is in unknown state, detail in the log file***
-300	Unexpected error
-500	Internal application error – from dll
-501	Internal application error – from terminal

Description of log format

The GpeEcrCommLib.dll created the logs with the internal process and communication between the terminal and dll. The log is the main part for resolving problems.

Definition of the configuration file

The GpeEcrCommLib needs to be configurable by the configuration file which is placed in the program folder. The location of the conf. file is handed over by calling API function *PosLib_GpeOpenLibrary ().* There is no possible to configure the dll directly through the API function at the run-time. The best way is placed conf. file into the same folder as the GpeEcrCommLib.dll

Supported operating system

The library was developed and tested on:

- Windows 7 Professional 32Bit/64Bit, SP 1
- Windows XP Professional 32Bit, SP 3

Known problems

Conclusion

GPE disclaims liability for errors in the examples and managed open source parts. If you find any error or mistake in documentation, please contact GPE APV POS department.

FAQ

This section will be made up of the frequent questions of ECR integrators.