



Document Description

**Baxi.net**  
**Application Developers Guide**

Version: 1.4.2.1

Date: 23.05.2016

## CONTENTS

	Baxi.net
	Application Developers Guide
	Version: 1.4.2.1
	Date: 23.05.2016
1	References
2	Revision Record
3	Introduction and environments
	Supported Microsoft .NET Frameworks on Windows:
	Recommended Windows OS versions:
	Supported Windows OS versions
4	Procedures for using BAXI
	4.1 Installation
	4.2 General work flow
	4.3 Error situations
	4.4 USB rs232 support
	4.5 Multi-instance support
	4.6 Bluetooth support
5	BAXI Properties
	5.1 Read properties
	5.2 Write properties
6	BAXI Methods
	6.1 General note on parameters.
	6.2 Open
	6.3 Close
	6.4 TransferAmount
	6.4.1 TransferAmount OptionalData Json fields
	6.4.1.1 autodcc
	6.4.1.2 merch
	6.4.1.3 txnref
	6.4.2 Usage of Pre-Authorisation in TransferAmount
	6.5 Administration
	6.5.1 Administration OptionalData Json fields
	6.5.1.1 merch
	6.5.1.2 txnref
	6.6 TransferCardData
	6.7 SendTLD
	6.8 GetTLDTag
	6.9 SendJson
	6.10 BiBAAdministration
	6.11 BiBTransaction
	6.12 GetFirstPairedCompanionAddress
	6.13 ActivateCompanion
7.	BAXI Events
	7.1 General note on events
	7.2 OnDisplayText
	7.3 OnPrinterText
	7.4 OnLocalMode
	7.5 OnError
	7.6 OnTLDReceived
	7.7 OnJsonReceived
	7.8 OnStdRsp
	7.9 OnTerminalReady
	7.10 OnLastFinancialResult
8	Baxi.ini file
9	Common Use Cases
	9.1 Transactions
	9.1.1 Purchase
	9.1.2 Return of goods
	9.1.3 Reversal
	9.1.4 Purchase with Cashback
	9.1.5 Balance inquiry
	9.1.6 Deposit
	9.1.7 Cash withdrawal
	9.1.8 Force Offline
	9.1.9 Pre-Authorisation
	9.2 TopUp
	9.3 Gift Card
	9.4 Group ID
	9.5 Get encrypted card data
	9.6 Get encrypted card data and issuer ID
	9.7 Send a track 2 card swipe to the terminal:
	9.8 TIP

- 9.9 Check host connection
- 9.10 Set terminal language
- 9.11 Receive a card inserted status event
- 9.12 "Get Customer Info"
- 9.13 "Get Customer Info" with Admin Finish.
- 9.14 Investigating results of last financial transaction.
- 9.15 JSON replacing TLD.
  - 9.15.1 CardInfo RequestAllTags(3999) using sendJson
  - 9.15.2 sendTld
  - 9.15.3 sendJson
- 9.16 JSON and Multiterminal
- 9.17 JSON and VAS (Value Added Services)
- 9.18 JSON and Terminal as an ECR Printer
- 10 MultiCurrency functionality
  - 10.1 Introduction & Setup
  - 10.2 Testing multicurrency
- 11 Best practice implementation of security in the ECR interface.
- 12 Deprecated Functions
- 13 Removed Functions
  - 13.1 TransferAmount\_V2
  - 13.2 TransferAmount\_V3
  - 13.3 TransferAmount\_V4
  - 13.4 Administration( int admCode, string OperId)
  - 13.5 SendTLD( string typeString, byte[] tldField)
  - 13.6 TransferCardData( int trackType, string trackData)
  - 13.7 BiBAdministration( int AdmCode, string OperID)
  - 13.8 BiBTransaction( int amount, string transactionData)
- 14 Appendices
  - 14.1 Appendix A – Status codes
    - 14.1.1 Error codes
    - 14.1.2 Method reject codes
  - 14.2 Appendix B – Sequence diagrams
    - 14.2.1 Initialisation
    - 14.2.2 Successful purchase
    - 14.2.3 Unsuccessful purchase
    - 14.2.4 Purchase with an error situation
  - 14.3 Appendix C – ECR Print Request\Response

## **1   References**

- Reference 1: Nets proprietary TLD (TypeLengthData) format description
- Reference 2: BiBOpos

## 2 Revision Record

DATE	VER.	COMMENTS	Author
22.12.2010	1.0.0.16	Version step only.	
14.01.2011	1.0.0.17	Version increment only.	
19.01.2011	1.0.2.0	Version increment only. See release notes for more details.	
23.02.2011	1.0.2.1	Added error code 7104.	
24.02.2011	1.0.2.2	Version increment only. See release notes for more details.	
08.03.2011	1.0.2.3	USB monitoring/support, added chapter 4.4	
09.03.2011	1.0.2.4	Version step only.	
11.03.2011	1.0.2.5	Force Offline	
16.03.2011	1.0.2.6	Version step only	
21.03.2011	1.0.2.7	Version step only.	
22.03.2011	1.0.2.8	Version step only.	
01.04.2011	1.0.2.9	Version step. Added revision to enable documentation changes when BAXI version number should not change.	
12.04.2011	1.0.3.0	Version step only.	
15.04.2011	1.03.1	Internal release only	
19.04.2011	1.0.3.2	Version step only.	
23.05.2011	1.2.0.3	New interface. This involves refactoring existing implementations. It is especially important for COM clients to rebuild their interface files. Upgraded to support .net micro framework version 4.1.	
06.06.2011	1.2.0.5	Version step only.	
23.06.2011	1.2.0.6	Added Extended LcalMode Documenation: TotalAmount, RejectionSource, RejectionReason, TipAmount, SurchargeAmount, TerminalID, AcquirerMerchantID, CardIssuerName, ResponseCode, TCC, AID, TVR, TSI, ATC, AED, IAC	
18.07.2011	1.2.0.7	Added rejection error codes for new properties: UseDisplayTextID UseExtendedLocalMode	
11.08.2011	1.2.0.7-B	Added Aministration Finish.	
06.09.2011	1.2.0.8	Version step only.	
16.09.2011	1.2.0.9	Added new property SerialDriver. See properties chapter for more detail.	
03.10.2011	1.2.1.0	Version step only.	
17.10.2011	1.2.1.1	Version step only.	
21.10.2011	1.2.1.1-B	Added property DeviceString to document.	
02.12.2011	1.3.0.1	Added new admin messages for last transection receipt and result, as well as the last transaction result message.	

24.01.2012	1.3.0.2	Added new TLD fields:TCC,BankAgent Added new fields in extended Local Mode: AccountType,EncryptedPAN, BankAgent, organisationNumber.	
16.02.2012	1.3.0.3	Added DoNotSplitDisplayText in ini file.	
17.02.2012	1.3.0.4	Renamed DoNotSplitDisplayText UseSplitDisplayText	
22.03.2012	1.3.0.5	Renamed UseSplitDisplayText UseJoinDisplayText	
24.09.2012	1.3.0.6	Added property TerminalSwVersion.	
15.10.2012	1.3.0.7	Version increment only.	
18.12.2012	1.3.0.7-C	Added Admin codes for Dataset and Software download.	
14.01.2012	1.3.0.8-A	Version increment only.	
26.04.2013	1.3.0.9	Version increment only.	
15.05.2013	1.3.1.0	Version increment and update of event handling.  Remark! The event is fired from an internal thread in baxi. The ECR client code is therefore executed from the baxi thread.  It is important to release the event handler as fast as possible to let the thread go back and wait for more events.  We recommend that the ECR programmer just create a new event and send the information to an internal thread of the ECR app.	
24.05.2013	1.3.1.1	Version increment.  Added OptionalData in Transfer Amount and LocalMode.	
17.06.2013	1.3.1.2	Version increment. Added PreAuth Changed OptionalData in Transfer Amount and LocalMode.	
02.07.2013	1.3.1.3	Version increment. Added new error codes 7515 and 7516	
01.08.2013	1.3.1.4	Version increment.	
13.08.2013	1.3.1.5	Version increment. Added new error codes 7427	
23.08.2013	1.3.1.6	Version increment.	
04.09.2013	1.3.1.7	Version increment.	
21.10.2013	1.3.1.8	<i>Added missing documentation in section 4.6 Write properties:</i>  UseSplitDisplayText  LogAutoDeleteDays  SocketListener  SocketListenerPort  Use2KBuffer DisplayTextInLocalMode	
05.11.2013	1.3.1.9 A	Version increment.	
27.11.2013	1.3.1.9 B	Added JSON documentation	
16.01.2014	1.3.2.0	Version increment.	
24.02.2014	1.3.2.1	Added new error codes 2012	
23.04.2014	1.3.2.1 C	Added Autodcc documentation	
22.05.2014	1.3.2.1 D	Added Multiuser merch documentation	
10.06.2014	1.3.2.2	Version increment.	
10.07.2014	1.3.2.3	Version increment.	
23.07.2014	1.3.2.4	Version increment and added new error code 7016	
09.10.2014	1.3.2.5	Added new write property ClientID Added documentation on how to setup multiple instances of baxi	

02.12.2014	1.3.2.6	Bugfixes Added new Method reject codes 7601, 7602 and 7603 Added new write properties UseMultiInstance and MultiInstanceConfigFile	
19.01.2015	1.3.2.7	Bugfixes Implemented Multiuser JSON support and added documentation for the same. Added correct JSON examples to the testgui.	
21.05.2015	1.3.2.9	Added new API functions GetFirstPairedCompanionAddress and ActivateCompanion.	
27.10.2015	1.3.2.10	<ul style="list-style-type: none"> <li>Renamed DoNotSplitDisplayText to UseSplitDisplayText in "Method reject codes" documentation.</li> <li>Changed the description for "TraceLevel" in section "BAXI Properties".</li> <li>Changed Bluetooth support section.</li> </ul>	
13.11.2015	1.3.2.11	Changed Bluetooth support section.	
12.01.2016	1.4.0.4	<ul style="list-style-type: none"> <li>Added transactiontype Bonus</li> </ul>	
30.03.2016	1.4.2.0	<ul style="list-style-type: none"> <li>Added Terminal as an ECR Printer.</li> <li>Added Appendix C: ECR Print Request\Response in documentation.</li> <li>Added multicurrency functionality</li> </ul>	
11.05.2016	1.4.2.0	Corrected outdated info about supported environments and a few other minor updates	AKa
23.05.2016	1.4.2.1	.net Framework 4.0 is also supported.	AKa

### 3 Introduction and environments

Baxi library is to simplify the integration of EFTPOS solutions to Windows-based point of sale (POS/ECR) applications. Baxi is a .NET library (DLL) and can be used with tools having support for .NET framework.

#### Supported Microsoft .NET Frameworks on Windows:

- .NET 3.5
- .NET 4.0
- .NET 4.5

#### Recommended Windows OS versions:

- Windows7
- Windows10

#### Supported Windows OS versions

- Windows XP
- Windows7
- Windows8/8.1
- Windows10

*Note! Library has not been tested with Mono framework and Nets do not currently support such environments.*

#### Recommended terminal software:

- Viking software version 04.70
- *Note! Baxi is backwards compatible with older Viking software versions but some of the new functionality may not work as expected!*



## 4 Procedures for using BAXI

Baxi can be used with multiple instances of baxi in the same process space.

It is important to setup the following write properties to unique values to avoid using the same external resources.

- Baxi.ClientID
- Baxi.LogFilePrefix
- Baxi.ComPort
- Baxi.SocketListener
- Baxi.SocketListenerPort

BAXI has four types of communication channels:

- *Methods* that can be called from the application to typically start a transaction
- *Events* that are sent from the terminal
- *Properties* containing information that can be read or written.
- *Init file* with configuration parameters

BAXI has two major states to identify the transaction process - bank mode and local mode.

- In bank mode, the terminal is in control and processing transaction
- In local mode, the ECR is in control and transaction has ended in the terminal
  - Local mode response will tell the result of the action

Any method that puts the terminal in bank mode will always result in a local mode event.

The local mode event usually indicates the final result of the operation.

### 4.1 Installation

Installation package for the Baxi.net does not include installer and therefore installation is done manually. Baxi\_dotnet.dll file is placed in the same directory as the application executable and linked to the application. Nets do not install library to Global Assembly Cache.

### 4.2 General work flow

1. Call a BAXI method.
2. Wait for incoming events from BAXI.
3. When an event gets triggered, act on the parameters of the event function.
4. The *OnLocalMode* event signals a finished transaction.

### 4.3 Error situations

An error event normally occurs if a selected function cannot be performed.

When BAXI has lost communication with the terminal during a transaction, the *OnLocalMode* event will be triggered, with the local mode result set to an unknown status.

The status of the transaction must then be investigated!

This can either be done manually, or new in version 1.3.0.1, via the ECR admin messages "Get Latest Financial Transaction Result" and "Get Latest Transaction Receipt".

These messages are documented in Chapter 5 and 6.

### 4.4 USB rs232 support

BAXI.net supports monitoring of USB rs232 devices.

This is to allow hot swapping USB payment terminals and that USB payment terminal reboots and still has contact with BAXI.

USB rs232 support is always enabled, but its behaviour is controlled by the ComPort and DeviceString properties.

- **PSI:**  
The Microsoft serial port implementation in the .net framework has a serious flaw with regards to USB rs232 ports disappearing. It is imperative that USB rs232 solutions use the Nets serial port driver. See the "SerialDriver" property for more details.
- **ComPort:**  
If this property is set to 0, this means that only DeviceString is used to dynamically find and monitor a USB rs232 com port that matches the DeviceString.

Multiple payment terminals on one machine is not supported.

If ComPort is set to a number and DeviceString has a value, BAXI will first try this com port number. If it fails to open this com port, it will dynamically find any USB port based on the value in DeviceString.

If DeviceString is not set, it will only use and monitor the port set in ComPort.

- **DeviceString:**

This property matches the device friendly name that is part of the COM port name in Windows' Device Manager.

For example, for the Ingenico Telium2 terminals, they are currently identified as DeviceString="SAGEM MONETEL USB Telium".

This device friendly name can also be found in the registry under the USB enumerations, found under

HKEY\_LOCAL\_MACHINE/SYSTEM/CurrentControlSet/Enum/USB

Under this key, all vendors have their own vendor and product ID. For Example, the Ingenico Telium 2 terminals are

Vid\_079b&Pid\_0028.

All entries will be under here and contain the following....

Navn	Type	Data
(Standard)	REG_SZ	(verdi ikke angitt)
Capabilities	REG_DWORD	0x00000084 (132)
Class	REG_SZ	Ports
ClassGUID	REG_SZ	{4D36E978-E325-11CE-BFC1-08002BE10318}
CompatibleIDs	REG_MULTI_SZ	USB\Class_02&SubClass_02&Prot_01 USB\Class_02&SubClass_02 USB\Class_02
ConfigFlags	REG_DWORD	0x00000000 (0)
DeviceDesc	REG_SZ	SAGEM MONETEL USB Telium
Driver	REG_SZ	{4D36E978-E325-11CE-BFC1-08002BE10318}\0011
FriendlyName	REG_SZ	SAGEM MONETEL USB Telium (COM11)
HardwareID	REG_MULTI_SZ	USB\vid_079b&pid_0028&rev_0000 USB\vid_079b&pid_0028
LocationInformation	REG_SZ	USB Device
Mfg	REG_SZ	SAGEM MONETEL T
Service	REG_SZ	usbser
UINumber	REG_DWORD	0x00000000 (0)

Use Friendly Name or DeviceDesc as identifier in baxi.ini and do not include the (COMXX) part.

Below are the scenarios for USB support in baxi.ini described in detail.

1. **ComPort set to X, DeviceString not set:** BAXI will attempt to open Com X.  
If that fails, BAXI reports an error.  
If it succeeds, BAXI will only monitor inserts and removals of Com port X only.
2. **ComPort set to X, DeviceString set:** BAXI will first attempt to open Com X.  
If that fails, BAXI will dynamically try to find and open any com port that matches DeviceString.  
BAXI will monitor all com ports that match DeviceString for inserts/removals.
3. **ComPort set to 0, DeviceString set:** In this case, BAXI will skip straight to trying to find a com port dynamically, based on DeviceString.  
BAXI will monitor all com ports that match DeviceString for inserts/removals.

Default is ComPort set to 0 and DeviceString set.

## 4.5 Multi-instance support

BAXI.net supports running in a multi-instance environment.

This means that several BAXI instances can be run in parallel.

To be sure that everything is working correctly, the write property UseMultiInstance should be set to 1. Otherwise, 2 major errors can happen.

1. Logging could be haltered. Since if multi-instance environment is not turned on then logging could be done by several instances to the same file. This could mean that all log lines are not written to the rightful files.
2. Several clients could have the same client ID, which could have very bad consequences.

If BAXI is run in a multi-instance environment and the UseMultiInstance setting is set to 1, then BAXI will add the Client ID to the name of the log files, making them unique for that instance and therefore avoiding problem one.

Also, if this setting is set, then BAXI will also look for the setting MultiInstanceConfigFile, this setting will point out a file somewhere on the file system that will be used by BAXI to coordinate all different instances of BAXI running on this machine.

Basically, each BAXI instance will write down its Client ID to this file and no one of them will allow an instance using a name that already exists.

This path does not have to be pre-created, if it does not exist at open, BAXI will create it.

## 4.6 Bluetooth support

BAXI.net supports connecting bluetooth payment terminal - also called companion - directly.

Companion needs to be paired with Windows and settings BluetoothTunnel and SocketListener needs to be set to 1. After companion is paired with Windows, it needs to be activated using ActivateCompanion function.

Using this function requires administrator rights.

Activation needs to be done only once.

Bluetooth support requires three additional DLLs: PclUtilities.dll, PCLService.dll and dllsign.dll.

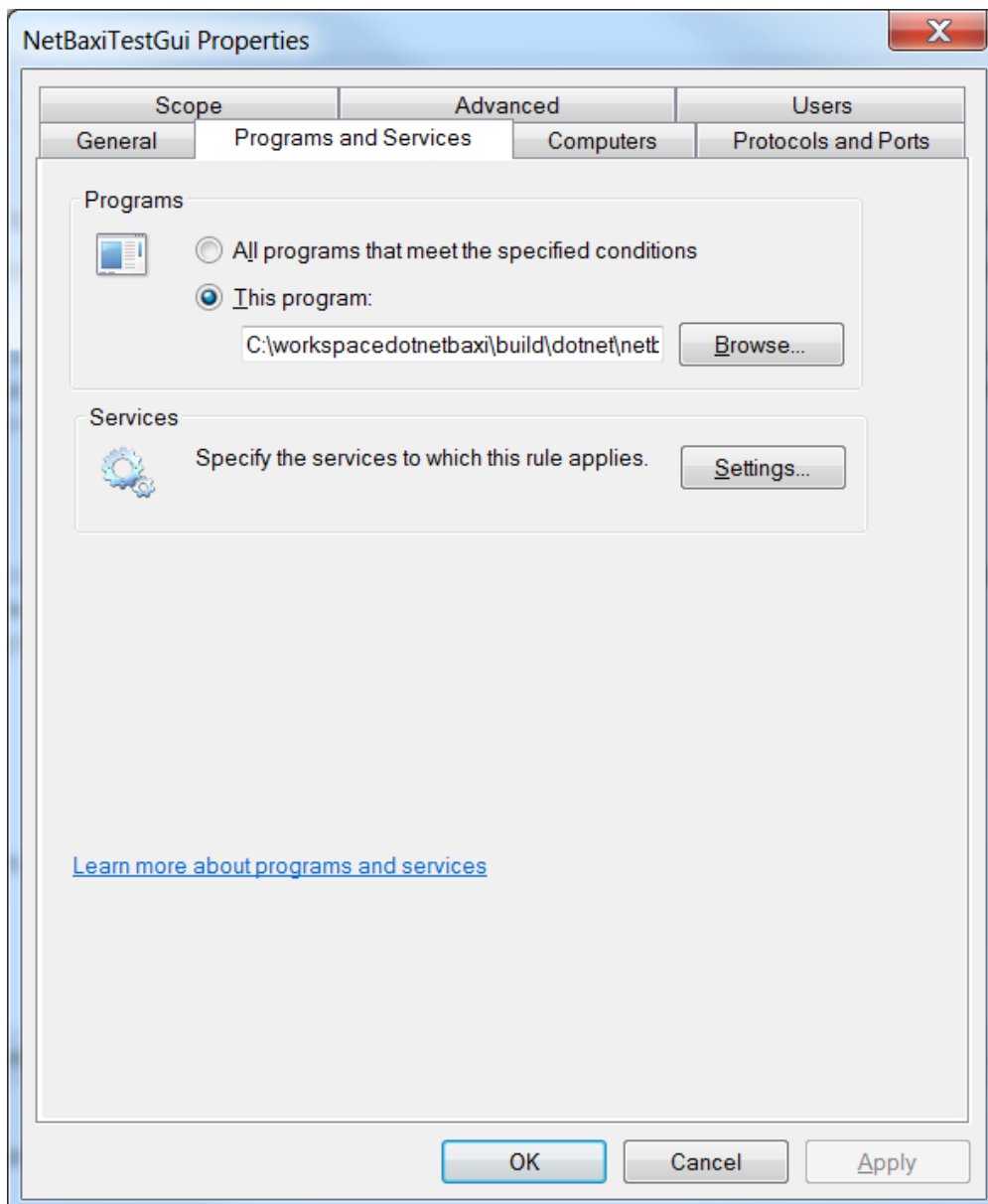
There are three different sets of these DLLs.

- Windows 7/8 32 bit
- Windows 7/8 64 bit
- Windows XP 32 bit

Depending used Windows environment, corresponding DLLs should be copied to Baxi working folder.

*Note! "Visual c++ 2012 redistributable x64" is required to be installed, to get the PCLUtilities.dll working properly. (Or x86 if applicable)*

*Note! Windows firewall "inbound rule" is required to be setup as shown below.*



**Note :** Make sure that program name will not contain any windows variable(i.e. Path has to be absolute path starting from drive name).

NetBaxiTestGui Properties

Scope

Advanced

Users


General

Programs and Services

Computers

Protocols and Ports

Protocols and ports



Protocol type:

TCP

Protocol number:

6

Local port

All Ports

Remote port

All Ports

Example: 80, 443, 5000-5010

Example: 80, 443, 5000-5010

Internet Control Message Protocol (ICMP) settings:

Customize...

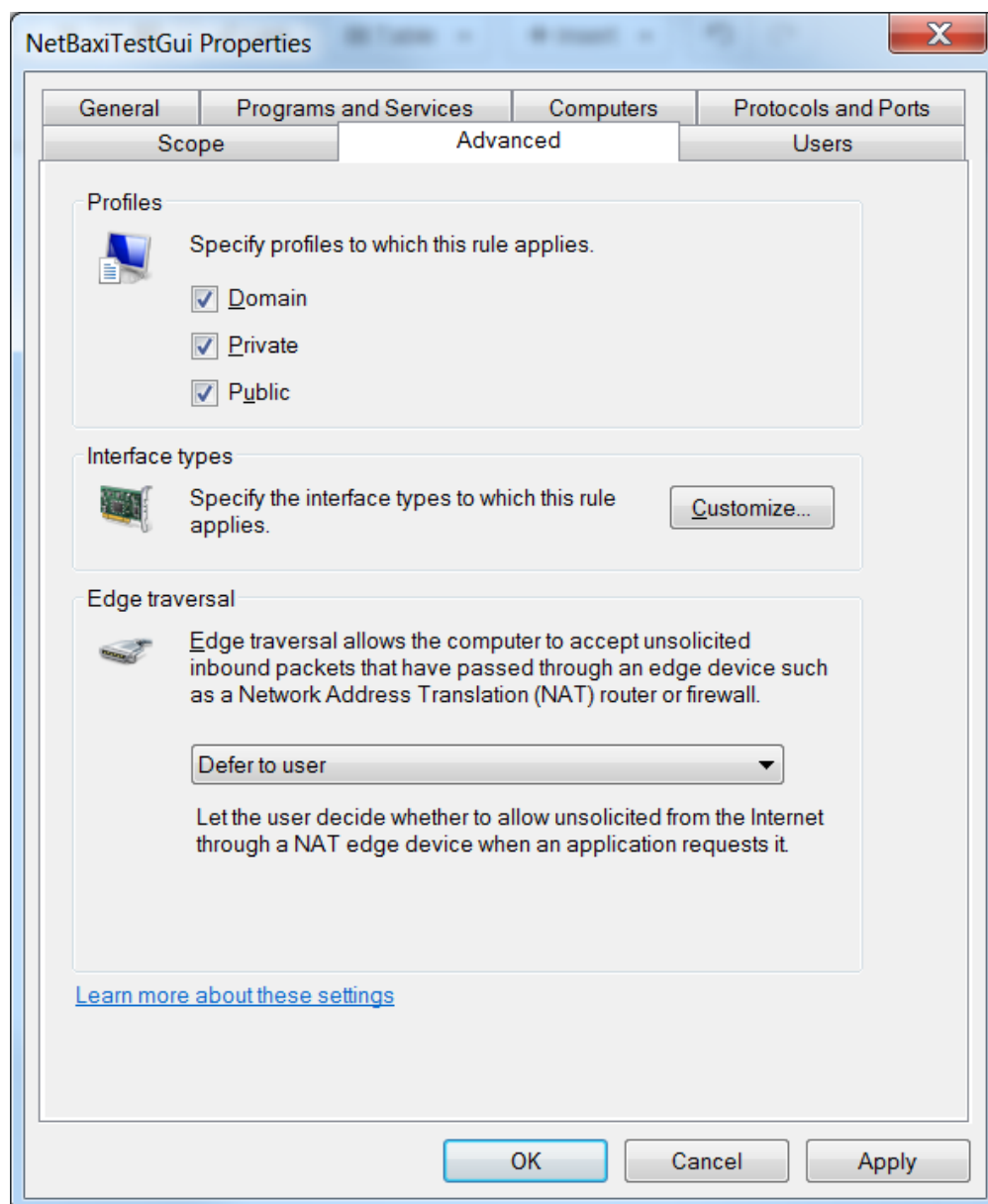
[Learn more about protocol and ports](#)

OK

Cancel

Apply

Application is required to have the access to all TCP ports.



Allow it for all profiles.

## 5 BAXI Properties

This list describes the properties that can be read /written in BAXI.

Write properties (I/O = W) must be written before the method Open() is called.

Any write property corresponding to a tag in the init file will override the init file tag value.

### 5.1 Read properties

These are available after a successful Open() dialogue.

Name	Type	I/O	Description
Version	string	R	This property is the BAXI version number. The format is a string of at least 35 in length containing a detailed version string for the BAXI component.
TerminalType (old name Term_sw_version)	string	R	This property is: First 2 bytes terminal type. Second 2 bytes Sofie protocol version. It can be read after a successful Open dialogue.
TerminalID	string	R	This property is the terminal ID. It can be read after a successful Open dialogue.
TerminalSwVersion	String	R	This is the Terminal software version, corresponding to the "ITU SW Version" tag 1005.
TerminalDeviceData_TLD	byte[]	R	This property contains TLD tag info retrieved from the terminal at start up. See Ref. 1 for details
MethodRejectInfo	string	R	This property contains a string describing the reason why the last method call has returned 0 (rejected a request).
MethodRejectCode	int	R	This property contains a code describing the reason why the last method call has returned 0 (rejected a request). A list of method reject codes is available in Appendix A.

### 5.2 Write properties

These properties are also found in the baxi.ini file and can be both written to and read.

All have default values in the baxi.ini file.

Setting a property will override the corresponding entry in the baxi.ini file.

If all entries in the baxi.ini file are set by properties, the baxi.ini file will be redundant and can be removed.

Name	Type	I/O	Description
LogFilePrefix	string	R+W	A string which will be the prefix of the log file name. The log file name will end with the current date.
LogFilePath	string	R+W	The path to the wanted log directory.
TraceLevel	int	R+W	Accepted values are valid trace levels. Ex. 0 = OFF, 1 = Minimal (Error), 2 = Low (Error + Trace), 3 = Medium (Error + Trace + Debug)
BaudRate	int	R+W	Accepted values are valid baudrates. Ex 9600.
ComPort	int	R+W	Accepted values are com ports (1 to 9). Ex 1.
HostPort	int	R+W	Accepted value is an int in the normal TCP/IP port number range (0 to 65536).
HostIpAddress	string	R+W	Accepted value is a string containing an ip address, e.g. 192.168.1.1
MsgRouterOn	short	R+W	Accepted values are 0 or 1. 0 means no Msg router used.
MsgRouterIpAddress	string	R+W	This property sets the parameter "IpAddress" from the ini-file. Accepted value is a string containing an ip address, e.g. 192.168.1.1
MsgRouterPort	short	R+W	This property sets the parameter "Port" from the ini-file. Accepted value is an int in the normal TCP/IP port number range (0 to 65536).
IndicateEotTransaction	int	R+W	Accepted values are 0 or 1. 0 means no EOT indication.
CutterSupport	int	R+W	Accepted values are 0 or 1. 0 means no cutter support.

PrinterWidth	int	R+W	This property is in use only if the application prints the content of the PrintLine property. Default value is 60. In BAXI, assigning the value 0 to the PrinterWidth property makes the terminal do all the printing locally on the built in printer.
DisplayWidth	int	R+W	Accepted values are 10 through 99.
VendorInfoExtended	string	R+W	<p>This property contains four fields.</p> <ul style="list-style-type: none"> <li>Vendor Name, this will be a 3 letter description supplied from Nets to each vendor. Fixed length, 3 letters (A-Z). <i>Please contact your local support at Nets to agree unique code!</i></li> <li>Vendor product/Application name.</li> <li>Vendor version/Application version.</li> <li>Tell ID, which should be used as a unique id for the POS, TVM, shop or similar that the terminal is installed in. To be able to identify where the terminal is installed is a demand in several markets, so the system supplier should be able to track the terminal based on this information.</li> </ul> <p>Rules:</p> <ol style="list-style-type: none"> <li>Each field is terminated by the delimiter ";". The delimiter is mandatory. (Always 4 delimiters in string).</li> <li>Maximum total length is 32 characters, including the delimiters.</li> <li>All fields except VendorName have variable length and must be printable ASCII.</li> </ol> <p>Sample:</p> <ul style="list-style-type: none"> <li>BBS;Retail;02.13.01;57807343803;</li> </ul>
PowerCycleCheck	int	R+W	Checks if the attached terminal has rebooted and if so, gives a warning and aborts the current transaction. The warning will be in the form of an OnDisplayText event and an OnPrintText event.
TidSupervision	int	R+W	<p>Checks if the attached terminal has changed terminal id.</p> <p>Locks BAXI if TID mismatch! No transactions are allowed while BAXI is locked! A user trying to access a locked BAXI will receive an OnDisplayText event and an OnPrintText event with information about the locked state.</p> <p>To unlock BAXI, remove the file "terminalid.txt", situated in the runtime executable directory.</p>
AutoGetCustomerInfo	int	R+W	<p>Accepted values are 0 or 1. 1 means enabling of automatic Customer Info retrieval in the terminal.</p> <p><b>WARNING:</b> This functionality is only supported on terminal versions 07.1X and higher. It must be turned off for older releases.</p>
TerminalReady	int	R+W	<p>Accepted values are 0 or 1. 1 means enabling of terminal ready signalling.</p> <p><b>WARNING:</b> This functionality is only supported on terminal versions 07.1X and higher. It must be turned off for older releases.</p>
UseDisplayTextID	Int	R+W	Accepted values are 0 or 1. 1 means enabling usage of display text source identification and display text identification. See 6.2 for more details.
UseExtendedLocalMode	Int	R+W	Accepted values are 0 or 1. 1 means enable extended localmode. This means that the ECR will get new fields in localMode.
SerialDriver	String	R+W	<p>Current possible values are:</p> <ul style="list-style-type: none"> <li>Nets</li> <li>Microsoft</li> </ul> <p>The string is case sensitive. Default value if not set is "Microsoft".</p> <p>For any implementation relying on USB to rs232 comms, be it directly through the Telium USB interface, or through a USB to rs232 adapter, it is imperative to use the Nets driver. This is because the Microsoft implementation has a flaw in discovering a removal of such a device/com port. An exception will be thrown which is impossible to catch in BAXI, resulting in the application having to handle it or crash/shut down.</p> <p>The Nets serialport implementation supports USB rs232 solutions, as described in 4.4</p>
DeviceString	String	R+W	The string is case sensitive. See 4.4 for more details.
UseSplitDisplayText	Int	R+W	Accepted values are 0 or 1. 0 means to receive display text in single display text message towards ECR
LogAutoDeleteDays	Int	R+W	Accepted values are integer on how long we should save the Logging file in number of days.
BluetoothTunnel	Int	R+W	Accepted values are 0 or 1. 0 means bluetooth companion is not used. To use bluetooth companion, SocketListener needs to be set to 1, too.
SocketListener	Int	R+W	Accepted values are 0 or 1. 0 means don't setup a tcp server. 1 means setup a TCP-server on open and listen on SocketListenerPort.
SocketListenerPort	Int	R+W	This is the port number that the TCP-server will listen for Clients on.
Use2KBuffer	Int	R+W	Accepted values are 0 or 1. 0 means that the terminal shall use the original 1Kbyte buffer. 1 means use that the terminal shall have increased buffer capacity of above 2Kbyte.
DisplayTextInLocalMode	Int	R+W	Accepted values are 0 or 1. 1 means that the ECR system accepts display text in Local Mode
ClientID	String	R+W	This is used when running baxi in a multiinstance environment, to be able to separate running instances. Set the property to an meaningful name.
UseMultiInstance	Int	R+W	Accepted values are 0 or 1. 1 means that baxi is running in multiinstance environment.

MultiInstanceConfigFile	String	R+W	This is the path and name of the config file that will use when running baxi in multiinstance environment. Baxi will write clientids to this file.
-------------------------	--------	-----	--



## 6 BAXI Methods

Except `GetFirstPairedCompanionAddress` and `ActivateCompanion`, all other methods will return immediately with either 1 (success) or 0 (failure).

The return value indicates if the BAXI component has accepted the request.

A successful call that puts the terminal in bank mode will generate an *OnLocalMode* event and maybe other events as well.

See Appendix B for common sequence diagrams.

Methods that do not set the terminal in bank mode:

- `SendTLD()`
- `TransferCardData()`
- `Close()`

The parameter operator ID has two different meanings:

1. Single TID terminal - identify the operator initiating a transaction.
2. Multi TID terminal - identify the logical terminal ID that shall be used in the transaction. Legal values 1-10. 1 is the master TID.

If the parameter `OperID` is not used, it should be set to "0000".

### 6.1 General note on parameters.

Any method that expects parameters, will receive these in the form of an `XYZArgs` object.

For example, to call `TransferAmount`, a `TransferAmountArgs` object must be filled out and passed as a parameter to the `TransferAmount` method.

### 6.2 Open

*PARAMETER: NONE*

*RETURNS:*

- 1 on success, local mode will follow
- 0 on failure, no local mode

*USAGE:*

1. Call `Open()`. If `Open()` returns 0, check the `MethodRejectCode` to determine the cause of the failure
2. On success, wait for *OnLocalMode* and *OnError* event.
3. The result of *OnLocalMode* will determine if the dialogue was successful or not.

Initializes the control with the current values of the write properties and starts the communication with the terminal.

This method generates a device attribute dialogue with the terminal.

This method should be run at start up, after any write properties has been set.

### 6.3 Close

*PARAMETER: NONE*

*RETURNS:*

- 1 always (it never fails), no local mode.

*USAGE:*

1. Call `Close()`.
2. Shut down system if applicable.

This method performs shutdown of communication with the terminal.

Synchronous, returns when operation completed.

**IMPORTANT:**

This method shall be called prior to shutting down BAXI/ECR!

Neglecting to call `Close()` before shut down will result in resources not being released!

## 6.4 TransferAmount

PARAMETER: TransferAmountArgs args

RETURNS:

- 1 on success, local mode will follow
- 0 on failure, no local mode

USAGE:

1. Call *TransferAmount()*. If the function returns 0, check the *MethodRejectCode* to determine the cause of the failure.
2. On success, wait for *OnDisplayText*, *OnPrintText*, *OnLocalMode* and *OnError* events. Several *OnPrintText* events can occur during a transaction.
3. The result of *OnLocalMode* will determine if the transaction was successful or not.

TransferAmountArgs content:	
<i>operID:</i>	<i>string</i>
	See introduction.
<i>type1:</i>	<i>int</i>
	H30 = EFT Authorisation (KJØP) => Purchase amount H31 = Return of Goods (RETU) H32 = Reversal (ANNU) => Annulate last amount H33 = Purchase with Cash back (KONT) H34 = PRE Authorisation (GAR) H35 = Adjustment (OPPG) H36 = Balance Inquiry (DISP) H37 = Complete Receipt (KVIT) H38 = Deposit (INN) H39 = Cash Withdrawal (UT) H3A = Load e-purse card (LOAD) H3B = Merchandise(Topup) Purchase H3C = Merchandise(Topup) Reversal H3D = Merchandise(Topup) Correction H40 = Force Offline H41 = Incremental PRE Authorisation H42 = Reversal PRE Authorisation => Partial and complete H43 = Sale Completion PRE Authorisation
<i>amount1:</i>	<i>int</i>
	Total amount.
<i>type2:</i>	<i>int</i>
	H30 = not in use
<i>amount2:</i>	<i>int</i>
	Only used if <TYPE 1> = Purchase with Cashback (H33) – Total purchase amount.
<i>type3:</i>	<i>int</i>
	H30 = not in use H31 = not in use H32 = VAT (Value added tax) amount. The total tax amount supplied by the ECR.

<i>amount3:</i>	<i>int</i>
	VAT amount
<i>hostdata:</i>	<i>string</i>
	Variable field length, max 40 digits alphanumeric. The data is to be sent to the HOST. The data characters must be in range H20 to H7F. This field is optional.
<i>articleDetails:</i>	<i>string</i>
	Variable field, alphanumeric data. Each record is separated with H1e, ASCII RS, A maximum of 3 records can be sent in one message, each separated with RS. The field is used to identify dedicated articles or reference to articles. This field is optional. ART # must be filled with article numbers in the purchase transaction ART # must be filled with references to the articles in a reversal transaction and correction transaction. This is application dependent, and must be clarified in each project. A Merchandise Reversal transaction and Merchandise Correction transaction can only include one article at a time.
<i>paymentCondition</i> <i>Code:</i>	<i>string</i>
	Variable field length, max 3 alphanumeric char.
<i>authCode:</i>	<i>string</i>
	Authorisation Code field. Variable field length, max 6 characters. Lower and upper case is allowed. Special characters like '@#£¤%&' are NOT allowed. This field is optional. The field is used for Force Offline transactions.
<i>OptionalData:</i>	<i>string</i>
	Variable field, alphanumeric data. Each record is Optional Data field. Variable field length, max 1024 ASCII characters. The context, contents and format of this field is specified and maintained by Nets Terminal IT. See ECR Chapter 13 Appendix C

### 6.4.1 TransferAmount OptionalData Json fields

This section gives further details on the OptionalData field in TransferAmount method.

OptionalData field can handle the following json tags.

#### 6.4.1.1 autodcc

This tag is to specify the currency preference in advance in a transaction type if terminal settings, card and transaction type supports currency conversion.

There are three possible values:

- 0: prompt cardholder to choose currency.
- 1: Auto select cardholder currency
- 2: Auto Select merchant currency

Optional field construction in dynamic currency conversion context:

```
{
  "od": {
    "ver": "1.01",
    "nets": {
      "ver": "1.00",
      "ch13": {
        "ver": "1.00",
        "ta": {
          "ver": "1.00",
          "o": {
            "ver": "1.00",
            "autodcc": 0
          }
        }
      }
    }
  }
}
```

#### Usecase autodcc 1:

The Customer has a card with cardholder currency in EUR. For the original purchase transaction the Customer was given a choice to pay either in EUR or SEK.

The Customer opted to proceed with the cardholder currency, so he paid in EUR.

Now for a refund transaction, the Merchant already knows that the original transaction was in cardholder currency, so for a refund, there is no need to prompt any DCC selection to the Customer. Instead go directly for DCC lookup.

#### Usecase autodcc 2:

The Customer has a card with cardholder currency EUR. For the original purchase transaction, the Customer was given a choice to either pay in EUR or SEK.

The Customer opted to proceed with the merchant currency, so he paid in SEK.

Now for a refund transaction, the Merchant already knows that the original transaction was in merchant currency, so for a refund, there is no need to go for DCC lookup.

### 6.4.1.2 merch

This tag is applicable only to multi terminal configuration to enable selection of multi-user by its BAX number.

This tag specifies merchant selection in a transaction request from ECR.

This is an alternative to the already existing operator id field for merchant selection from ECR.

```
{
  "od": {
    "ver": "1.01",
    "nets": {
      "ver": "1.00",
      "ch13": {
        "ver": "1.00",
        "ta": {
          "ver": "1.00",
          "o": {
            "ver": "1.00",
            "merch": 733300
          }
        }
      }
    }
  }
}
```

### 6.4.1.3 txnref

This tag assigns a Transaction Reference Number to the current transaction.

The same reference number can later be used for Administration Reversals by sending the same reference number used for a Purchase earlier, thus specifically Reversing that transaction.

```
{
  "od": {
    "ver": "1.01",
    "nets": {
      "ver": "1.00",
      "ch13": {
        "ver": "1.00",
        "ta": {
          "ver": "1.00",
          "o": {
            "ver": "1.00",
            "txnref": "123456789123"
          }
        }
      }
    }
  }
}
```

## 6.4.2 Usage of Pre-Authorisation in TransferAmount

The OptionalData field is also used when doing pre-authorisation.

The flow of pre-authorisation transactions is the following:

1. A TransferAmount with 0x34 (Authorisation) is initiated from the ECR. This will reserve an amount on the credit card at the issuer. This amount is an estimate of the final amount used later on in the Sale Completion as described in step 5.
2. The terminal will respond with a localmode that holds a JSON formatted token in the OptionalData field. This token should be entered in the OptionalData field when a future operation with this transaction is done. The data given in the OptionalData field looks something like this:

```
{
  "od": {
    "ver": "1.01",
    "preauth": {
      "ver": "1.0",
      "auth": {
        "ver": "1.0",
        "token": {
          "ver": "1.0",
          "t": "6y9Em4Qpx0lXGwaAdaaBg2wEQmY=",
          "e": "140831"
        }
      }
    },
    "data": {
      "ver": "1.0",
      "id": 3,
      "tpan": "457199*****5001",
      "ref": "947011 124590",
      "TCC": D@1,
      "resp": "00"
    },
    "receipt": {
      "ver": "1.0",
      "cdt": {
        "ver": "1.0",
        "prnstr": [
          "VISA ",
          "*****5001"
        ]
      }
    }
  }
}
```

3. \*Optional\* A TransferAmount with 0x41 (Increment Pre-authorisation) is initiated from the ECR. This will add an amount to an already authorised transaction. This means that the token received in step 2. must be added to the OptionalData field in the initiated TransferAmount, this to give the terminal information about which transaction the new amount should be added on.
4. \*Optional\* A TransferAmount with 0x42 (Reversal Pre-authorisation) is initiated from the ECR. This will reverse the previously authorised transaction. This cannot be applied after that a TransferAmount with 0x43 (Sale completion Pre-authorisation) has been sent from the ECR.
5. A TransferAmount with 0x43 (Sale Completion Pre-authorisation) is initiated from the ECR. This will complete the previously pre-authorised transaction. The token received in step 2 must be added in the OptionalData field of this TransferAmount.

## 6.5 Administration

PARAMETER: AdministrationArgs args

RETURNS:

- 1 on success, local mode will follow
- 0 on failure, no local mode

USAGE:

1. Call Administration(). If the function returns 0, check the MethodRejectCode to determine the cause of the failure.
2. On success, wait for OnDisplayText, OnPrintText, OnLocalMode and OnError events. Several OnPrintText events can occur during an administration dialogue.
3. The result of OnLocalMode will determine if the administrative function was successful or not.

Starts one of the administration dialogues.

TransferAmountArgs content:	
<i>admCode:</i>	<i>int</i>
	<p>0x3030 = not used</p> <p>.....</p> <p>0x3039 = not used</p> <p>0x3130 = RECONCILIATION function.</p> <p>0x3132 = Cancellation request. The terminal has the choice to ignore the request. The OnLocalMode result will determine the final status of the transaction.</p> <p>0x3134 = ANNUL from ECR should be mapped by ITU to perform REVERSAL transaction.</p> <p>0x3135 = Balance Inquiry.</p> <p>0x3136 = X-report.</p> <p>0x3137 = Z-report.</p> <p>0x3138 = Send Offline Transactions to HOST.</p> <p>0x313A = Print of stored EOT transactions.</p> <p>0x313B=Finish current dialogue (behaves as Cancel, but silently).</p> <p>0x313C = Print Latest Financial Transaction Receipt</p> <p>0x313D = Send Latest Financial Transaction Result</p> <p>0x313E = Start Software Download</p> <p>0x313F = Start Dataset Download</p>
<i>operID:</i>	<i>string</i>
	See introduction.

Messages 0x313C and 0x313D are used to find out the result the previous transaction.

This will typically be used when an error or loss of communications occurs and the result of the transaction is unknown.

By using these messages, one can find out if the ITU completed the transaction or not, so that a decision can be made to either attempt the transaction again or give the customer the proper receipt and goods.

0x313C gives the receipt copy as a printtext. 0x313D gives a Last Financial Transaction Result message (0x45) and its corresponding event in BAXI.

See Ch. 8 for a common usage example and Ch. 6 for description of the Last Financial Transaction Result event.

### 6.5.1 Administration OptionalData Json fields

This section gives further details on the OptionData field in the Administration method.

OptionalData field can handle the following json tags.

#### 6.5.1.1 merch

This tag is applicable only to multi terminal configuration to enable selection of multi-user by its BAX number.

This tag specifies merchant selection in a transaction request from ECR.

This is an alternative to the already existing operator id field for merchant selection from ECR.

```
{
  "od": {
    "ver": "1.01",
    "nets": {
      "ver": "1.00",
      "ch13": {
        "ver": "1.00",
        "ta": {
          "ver": "1.00",
          "o": {
            "ver": "1.00",
            "merch": 733300
          }
        }
      }
    }
  }
}
```

### 6.5.1.2 txnref

This tag is the Transaction Reference Number and is used for Administration Reversals by sending the same reference number used for an earlier Purchase, thus specifically Reversing that transaction.

```
{
  "od": {
    "ver": "1.01",
    "nets": {
      "ver": "1.00",
      "ch13": {
        "ver": "1.00",
        "ta": {
          "ver": "1.00",
          "o": {
            "ver": "1.00",
            "txnref": "123456789123"
          }
        }
      }
    }
  }
}
```

## 6.6 TransferCardData

PARAMETER: TransferCardDataArgs args

RETURNS:

- 1 on success, no local mode
- 0 on failure, no local mode

USAGE:

1. Call *TransferAmount()* to do a deposit.
2. Call *TransferCardData()*. If the function returns 0, check the *MethodRejectCode* to determine the cause of the failure.
3. On success follow the procedure of a normal transaction.

This method sends the information in a magnetic card track to the Terminal.



The terminal will act as if the card was swiped locally on the terminal.

The function is targeted at the specific use case of depositing money on gift cards.

It is not intended to be used to enable the ECR to pass customer card track contents to the terminal!

Only track 2 content cards are supported.

This message is only allowed when the terminal is in bank mode typically during a deposit.

TransferCardDataArgs content:	
<i>trackType:</i>	<i>int</i>
	0x32 = Origin track 2
	0x40 = Origin Manually
<i>trackData:</i>	<i>string</i>
	The track content.

## 6.7 SendTLD

PARAMETER: SendTldArgs args

RETURNS:

- 1 on success, sometimes local mode, depending on TLD tag range
- 0 on failure, no local mode

USAGE:

1. Call `SendTLD()` to perform a function. If the function returns 0, check the `MethodRejectCode` to determine the cause of the failure.
2. On success and if `typeString` is set to "REQ", wait for the response in the `OnTLDReceived` event.

This method is a general purpose method, used to trigger various functions.

This function will set the protocol in bank mode, thus end the dialogue with a local mode message when using TLD tags in the groups "Card info" (range 2000) and "ECR info" (range 3000).

Currently, the function does not set the protocol in bank mode for requests in the 1000 range.

SendTldArgs content:	
<i>TldType:</i>	<i>string</i>
	Specifies the TLD tag type.  Possible values are: <ul style="list-style-type: none"> <li>• "REQ"</li> <li>• "CMD"</li> </ul> 0x40 = Origin Manually
<i>TldField:</i>	<i>byte[] (array of bytes)</i>
	A TLD field built up as described in ref 1.

## 6.8 GetTLDTag

PARAMETER: byte[] wantedTldTag, byte[] tldBuffer, byte[] wantedTldTagValue

RETURNS:

- 1 on success, no local mode
- 0 on failure, no local mode

USAGE:

1. Call `GetTLDTag()` to retrieve a wanted tag value from a TLD buffer.

This method is a helper function for parsing a TLD buffer.

It is utilitarian in nature and does not use the `EventArgs` interface contract, like the other methods.

<i>wantedTldTag:</i>	The wanted TLD tag.
<i>tldBuffer:</i>	The TLD buffer in which to search.
<i>wantedTldTagValue:</i>	On success this byte buffer will hold the wanted tag value.

## 6.9 SendJson

PARAMETER: SendJsonArgs args

RETURNS:

- 1 on success
- 0 on failure

USAGE:

- Call *SendJson()* to perform a function.
- If the function returns 0, check the *MethodRejectCode* to determine the cause of the failure.
- On success Wait on the response in the *OnJsonReceived* event.

<i>SendJsonArgs</i> content:	<i>public string JsonData;</i>
------------------------------	--------------------------------

This is a general purpose method used to trigger various responses from either the terminal or the host.

JSON is preferred over TLD whenever applicable and adds more functionality then TLD.

Currently only TLD tag 3999 – CardInfo RequestAllTags – is covered by JSON but with every release of the Terminal Software more tags will be added to the JSON interface until it completely deprecates TLD.

Firstly *SendJson()* replaces *SendTld()* in this fashion.

Secondly it adds the ability to query the Host for various Value Added Services. Because of the flexibility of this interface Baxi will not need to be upgraded whenever a new such Value Added Service is provided, allowing for easy and quick integration in the ECR.

Thirdly, starting from Baxi.iOS v1.1.3.0 and later, it allows to send ECR print messages including text, barcode and bitmap to ITU so as to support the use of terminal as ECR printer feature.

See chapter 8 for example use cases & see Appendix C for the instruction of ECR print JSON message.

## 6.10 BiBAdministration

PARAMETER: BiBAdministrationArgs args

RETURNS:

- 1 on success, local mode will follow
- 0 on failure, no local mode

USAGE:

- Special custom function, see Ref [2]. Similar in usage to the *Administration()* method.

## 6.11 BiBTransaction

PARAMETER: BiBTransactionArgs args

RETURNS:

- 1 on success, local mode will follow
- 0 on failure, no local mode

USAGE:

- Special custom function, see Ref [2]. Similar in usage to the *TransferAmount* method.

## 6.12 GetFirstPairedCompanionAddress

PARAMETER: NONE

RETURNS:

- On success, companion's bluetooth address as string
- On failure, an empty string

## USAGE:

- Call *GetFirstPairedCompanionAddress()*.
- Functions searches all Ingenico bluetooth companions paired with Windows and returns the first found companions bluetooth address.

## 6.13 ActivateCompanion

*PARAMETER: string bluetoothAddress*

## RETURNS:

- 1 on success
- 0 on failure

## USAGE:

- Call *ActivateCompanion(<Bluetooth address>)*. Bluetooth address is max. 13 characters long string, e.g. "547F545948F1". If an empty string or "0" is given as a parameter, Baxi will select the first Ingenico bluetooth terminal paired with Windows which it finds.

Activates Ingenico companion paired with Windows.

It should be enough to call this fuction just once unless the selected companion wants to be changed.

After calling this function Windows remembers the last selected companion.

When this function has been called and in baxi.ini there is setting BluetoothTunnel=1 under section [TCIPSERVER], function Open will open connection to activated bluetooth companion .

## 7. BAXI Events

### 7.1 General note on events

All event delegates that contain data, will give this data in EventArgs objects.

For example, the OnDisplayText event, will give a DisplayEventArgs object that contains the display text string and any other pertinent fields as well.

Remark!

The event is fired from an internal thread in baxi.

The ECR client code is therefore executed from the baxi thread.

It is important to release the event handler as fast as possible to let the thread go back and wait for more events.

We recommend that the ECR programmer just create a new event and send the information to an internal thread of the ECR app.

### 7.2 OnDisplayText

```
public delegate void DisplayEventHandler(object sender, DisplayEventArgs e);
```

DisplayEventArgs contents:	
DisplayText:	string
	A display text has been received from the terminal. Max length is 80 characters.
DisplayTextSourceID:	int
	The source that generated the display text. A TLD field built up as described in ref 1.
DisplayTextID:	int
	The unique identifier of a display text. Mostly intended for unattended solutions. See 4.6 UseDisplayTextID.

### 7.3 OnPrinterText

```
public delegate void PrintEventHandler(object sender, PrintEventArgs args);
```

PrintEventArgs content:	
PrintText:	string
	<p>This is the printer text (the receipt) formatted in lines with a maximum line length of 24 characters. Several print text messages could come during a transaction. Separators:</p> <ul style="list-style-type: none"> <li>• 0xA = Separator line feed. Will be used throughout the printText string to allow correct formatting of the receipts.</li> <li>• 0xC = Separator end of text. Will be used at the very end of every string to indicate the end of text.</li> <li>• 0xE = Separator cutter(optional). Can appear if the property <i>CutterSupport</i> is set. Can be used anywhere in the printText string to indicate a request to cut the receipt.</li> </ul>

### 7.4 OnLocalMode

```
public delegate void LocalModeEventHandler(object sender, LocalModeEventArgs args);
```

Signals the application that a financial or administrative transaction is completed.

**LocalModeEventArgs content:**

<i>LocalModeResultData :</i>	<i>string</i>
	This parameter contains the complete local mode reply from the terminal in raw format.
<i>Result:</i>	<i>int</i>
	<p>This is the result of the transaction:</p> <ul style="list-style-type: none"> <li>• 0 : Financial transaction OK, accumulator updated</li> <li>• 1 : Administrative transaction OK, no update of accumulator</li> <li>• 2 : Transaction rejected, no update of accumulator</li> <li>• 3 : Transaction is Loyalty Transaction</li> <li>• 99 : Unknown result. Lost communication with terminal. Baxi has generated this local mode.</li> </ul>
<i>AccumulatorUpdate:</i>	<i>int</i>
	<p>H20 = indicates standard update of accumulator.  H21 = Not used in current software in the ITU.  H22 = indicates transaction is finalised as Offline transaction.  H23 .. H2F = Reserved for future use. The ECR shall finalise the transaction as OK. Accumulator type is uncertain for the time being.  H30 = indicates no update of accumulator.</p>
<i>IssuerId:</i>	<i>int</i>
	This is the card issuer ID.
<i>TruncatedPan:</i>	<i>string</i>
	This parameter contains card information. The format is the card number with length up to 19 digits.
<i>Timestamp:</i>	<i>string</i>
	<p>This parameter contains the timestamp of the transaction.  The format is YYYYMMDDHHMMSS</p>
<i>VerificationMethod:</i>	<i>int</i>
	<p>This is a code for cardholder verification method (CVM). The codes are:</p> <p>0: Transaction is PIN based (default)  1: Transaction is signature based  2: No CVM. With or without amount confirmation by cardholder.  3: Transaction is Loyalty Transaction  For the code 3, local mode result will always be 3.</p>
<i>TotalAmount:</i>	<i>int</i>
	<p>11bytes numeric data(H30.. H39).</p> <p>Used in:</p> <ul style="list-style-type: none"> <li>• Restaurant or Hotel environment where TIP is added to the purchase amount on the ITU. Used in the Purchase or Adjustment transaction.</li> <li>• Transactions with surcharge. This total amount will contain the sum of the original amount plus the surcharge amount.</li> </ul>
<i>SessionNumber:</i>	<i>string</i>
	This parameter contains the session number of the transaction (3 bytes numeric data). This number is global for a site and gets incremented after a reconciliation.
<i>StanAuth:</i>	<i>string</i>
	This parameter contains the transaction reference number. The length is 12 characters
<i>SequenceNumber:</i>	<i>string</i>
	4 bytes numeric data (H30 .. H39). This is the customer number if the transaction was Pre-Auth transaction. Must be used as reference in Transfer Amount - Adjustment transaction.

<i>TotalAmount:</i>	<i>int</i>
	This parameter is the Total ITU transaction amount used in Restaurant or Hotel environment where TIP is added to the purchase amount on the ITU. It's also used if a surcharge fee is added to the purchase amount on the ITU.
<i>RejectionSource:</i>	<i>int</i>
	This parameter indicates the source for the rejection. Fixed length, 2 characters. This field is optional. Rejection sources will be added as they are needed. If a rejection source is present, there will always also be a rejection reason present.
<i>RejectionReason:</i>	<i>string</i>
	This parameter indicates the reason for a rejection. Variable length, maximum 20 characters. This field is optional. Rejection reasons will be added as they are needed. If a rejection reason field is present, there will always also be a rejection source present.
<i>TipAmount:</i>	<i>int</i>
	This parameter is used in Restaurant or Hotel environment where TIP is added to the purchase amount on the ITU.
<i>SurchargeAmount:</i>	<i>int</i>
	This parameter is used if surcharge fee is added to the purchase amount on the ITU.
<i>TerminalID:</i>	<i>string</i>
	This parameter is the terminal ID.
<i>AcquirerMerchantID:</i>	<i>string</i>
	This parameter is the Site ID.
<i>CardIssuerName:</i>	<i>string</i>
	This parameter will be ApplicationLabel in case of chip cards
<i>ResponseCode:</i>	<i>string</i>
	This parameter will contain PSP response codes in case of online approved transactions & Y codes from terminal for offline approved transaction.
<i>TCC:</i>	<i>string</i>
	This parameter is Transaction Condition Code.3 byte alphanumeric data. Each byte indicates as following: TCC [0] = Card Data Entry Method TCC [1] = CVM Method TCC [2] = Authorization Method
<i>AID:</i>	<i>string</i>
	This parameter is Chip Card Application Identifier
<i>TVR:</i>	<i>string</i>
	This parameter is Chip Card Terminal Verification Result
<i>TSI:</i>	<i>string</i>
	This parameter is Chip Card Terminal Status Information
<i>ATC:</i>	<i>string</i>
	This parameter is Chip Card Application Transaction Counter

<i>AED:</i>	<i>string</i>
	This parameter is Chip Card Application Effective Data
<i>IAC:</i>	<i>string</i>
	This parameter is Chip Card Issuer Action Code
<i>OptionalData:</i>	<i>string</i>
	Variable field length, max 1024 ASCII characters. The context, contents and format of this field is specified and maintained by Nets Terminal IT. See ECR Chapter 13 Appendix C - Optional Data.

## 7.5 OnError

```
public delegate void BaxiErrorHandler(object sender, BaxiEventArgs args);
```

OnError signals to the application that an error has occurred.

BaxiEventArgs contents:	
<i>ErrorCode:</i>	<i>int</i>
	A list of error codes is available in Appendix A.
<i>ErrorMessage:</i>	<i>string</i>
	A short textual description of the error.

## 7.6 OnTLDRceived

```
public delegate void TLDReceivedHandler(object sender, TLDReceivedEventArgs args);
```

This event receive a TLD tag data string from the terminal. See ref 1 for details.

TLDReceivedEventArgs contents:	
<i>TldType:</i>	<i>string</i>
	Specifies the TLD tag type. Possible values are: <ul style="list-style-type: none"> <li>• "RESP"</li> <li>• "STATUS"</li> <li>• "INFO"</li> </ul>
<i>TldData:</i>	<i>byte[ ]</i> (array of bytes)
	A TLD field

## 7.7 OnJsonReceived

```
public delegate void OnJsonReceived(object sender, JsonReceivedEventArgs args);
```

This event receive a Json tag data string from the terminal. See ref.

JsonReceivedEventArgs contents:	
<i>JsonString:</i>	<i>string</i>

## 7.8 OnStdRsp

```
public delegate void StdRspReceivedHandler(object sender, StdRspReceivedEventArgs args);
```

This is an optional special purpose event. It triggers each time the terminal sends a response frame to BAXI.

A user wanting to make quick consecutive calls to BAXI can use this event to find out when BAXI has finished processing the last call.

Example of use:

1. TransferAmount();

2. Wait for OnStdRsp() event.
3. SendTLD();

StdRspReceivedArgs contents:	
Response:	string
	The response code from the terminal. Always "00" per today.

## 7.9 OnTerminalReady

*public delegate void TerminalReadyEventHandler(object sender, TerminalReadyEventArgs args);*

**WARNING:** This functionality is only supported on terminal versions 07.1X and higher. It must be turned off for older releases.

This is an optional special purpose event.

It triggers each time the terminal determines that it has completed terminal management.

Typically, this will occur as the terminal goes into idle after an activity, like a purchase.

After this event has triggered, the terminal should always be ready to handle a new function, such as TransferAmount.

One other use is for performing a Close-Open sequence without disrupting terminal communication towards host, such as swapping between two instances of BAXI.

TerminalReadyEventArgs contents:	
	There is no content so far for this EventArgs object.

## 7.10 OnLastFinancialResult

*public delegate void LastFinancialResultEventHandler( object sender, LastFinancialResultEventArgs args);*

This is a special purpose event.

It is sent by the ITU when the ECR invokes the message Admin(SendLatestFinancialResult(0x313D)).

The contents are identical to the contents of a Local Mode message, but it contains the results of the last/latest financial operation.

This data can be used to determine what happened to a transaction in case of communications errors or other causes of a loss of transaction result.



## **8 Baxi.ini file**

The baxi.ini file can be used to set up the environment for BAXI.

It should be placed in the same directory as the application executable running BAXI.

All the tag entries can also be set via properties. If there is a conflict between a tag entry and a property, the property always overrides the init-file.

If all the init-file tag entries are set by properties, baxi.ini is redundant and can be removed altogether.

The baxi.ini file entries maps 1-1 to the writable properties.

See corresponding write property for details!

## 9 Common Use Cases

Here follows a list of example usage of the current transaction types.

Basic functions that should always be integrated in a POS application are:

- Purchase
- Purchase with cash withdrawal
- Reversal
- Return of goods
- Reconciliation

It is important to implement reversal because it will be necessary to cancel a transaction if signature control fails or a similar situation.

US = 0x1F

RS = 0x1E

### 9.1 Transactions

#### 9.1.1 Purchase

Normal purchase transaction request(10.00 kr):

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x30;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

Normal purchase transaction request (10.00 kr) with VAT (3 kr):

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x30;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x32;
a.Amount3 = 300;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

Normal purchase transaction request (10.00 kr) with loyalty info ("abcdefg"):

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x30;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "abcdefg";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

Normal purchase transaction request (10.00 kr) with payment condition code "2":

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x30;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "2";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

### 9.1.2 Return of goods

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x31;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

### 9.1.3 Reversal

Reversal of the last transaction (10 kr).

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x32;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

### 9.1.4 Purchase with Cashback

Purchase transaction request (10.00 kr) with cash back (2 kr):

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x33;
a.Amount1 = 1200;
a.Type2 = 0x30;
a.Amount2 = 1000;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

### 9.1.5 Balance inquiry

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x36;
a.Amount1 = 0;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

### 9.1.6 Deposit

Can be used for example to deposit money on gift cards.

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x38;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

### 9.1.7 Cash withdrawal

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x39;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
```

### 9.1.8 Force Offline

Force an offline purchase of amount 1000 with authcode of 'ABC'.

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x40;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "ABC";
int result = Baxi.TransferAmount(a);
```

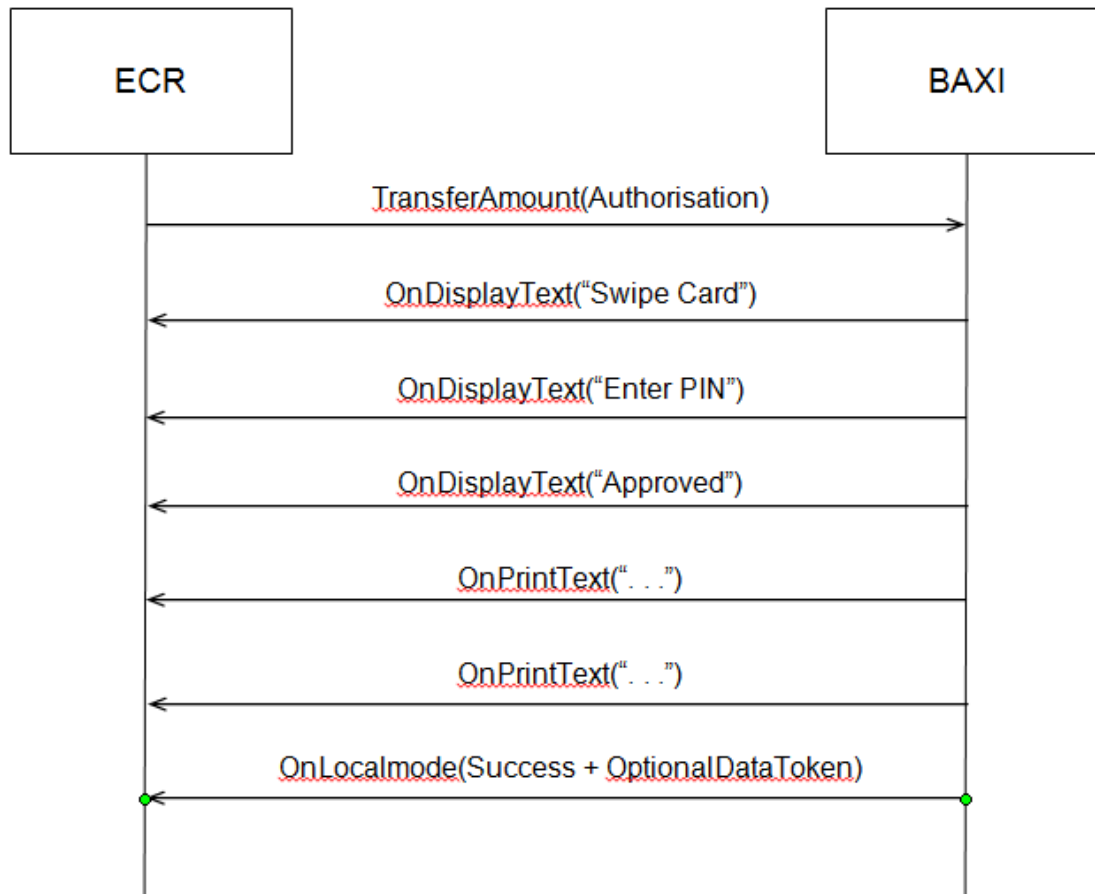
### 9.1.9 Pre-Authorisation

a) Done in at least two parts where the initial part Authorises an estimated amount, in this case Amount1=1000.

```
TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x34;
a.Amount1 = 1000;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "ABC";
int result = Baxi.TransferAmount(a);
```

In the OnLocalMode response keep what is sent in args.getOptionalData():

```
{
  "od": {
    "ver": "1.01",
    "preauth": {
      "ver": "1.0",
      "auth": {
        "ver": "1.0",
        "token": {
          "ver": "1.0",
          "t": "WZyq1XadTyso8AnpJdCTlGPKWUw=",
          "e": "140903"
        }
      }
    },
  },
  "data": {
    "ver": "1.0",
    "id": 4,
    "tpan": "521358****2083",
    "ref": "591504 125853",
    "TCC": "D@1",
    "resp": "00"
  },
  "receipt": {
    "ver": "1.0",
    "cdt": {
      "ver": "1.0",
      "prnstr": [
        "MASTERCARD ",
        "*****2083"
      ]
    }
  }
}
```



b) Use the OptionalData token from the original Authorisation to append any additional estimated Amount – in this example Amount1=500.

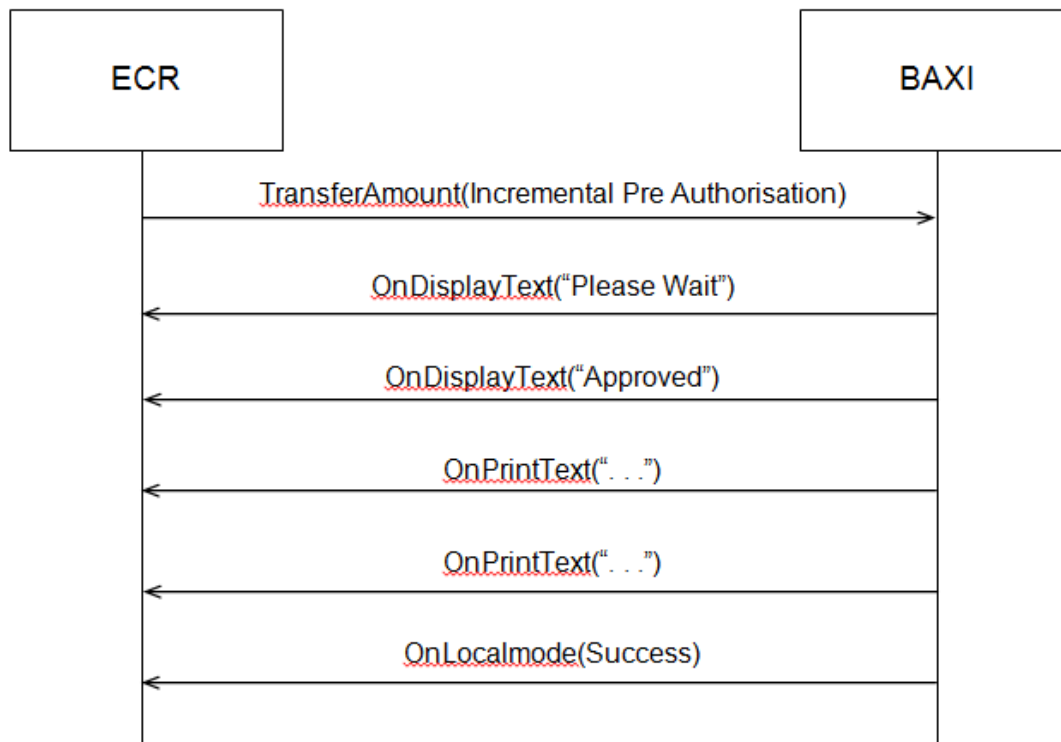
This doesn't require the card of the Customer. This is an optional step that is not needed to finish the transaction.

Escape any quotation marks in the middle of the OptionalData String.

```

TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x41;
a.Amount1 = 500;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "ABC";
a.OptionalData =
"{"od":{"ver":"1.01","preauth":{"ver":"1.0","auth":{"ver":"1.0","token":{"ver":"1.0","t":"WZyq1XadTyso8AnpJdCT1GPKWUw=", "e":"140903"}}}, "data":{"ver":"1.0","id":4,"tpan":"521358*****2083","ref":"591504125853","TCC":"D@1","resp":"00"},"receipt":{"ver":"1.0","cdt":{"ver":"1.0","prnstr":["MASTERCARD \", "*****2083"]}}}}";

int result = Baxi.TransferAmount(a);
  
```



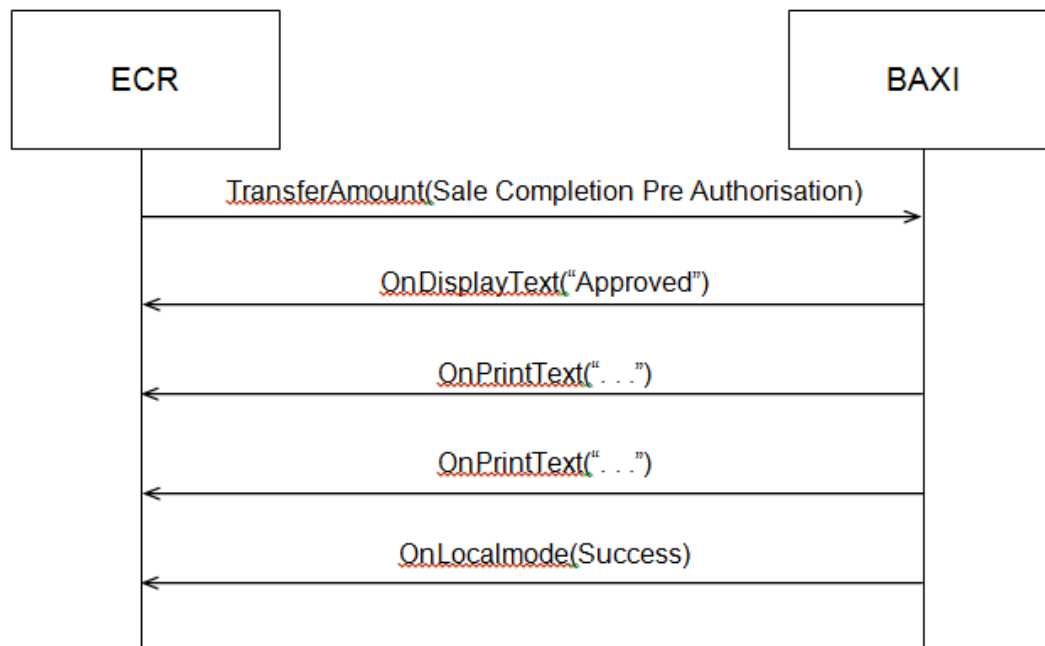
c) The final part is the Sale Completion. Please note that the final Amount may differ somewhat from the initially estimated Authorisation, in this case Amount1=1600.

This doesn't require the Customer's card.

```

TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x43;
a.Amount1 = 1600;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "";
a.PaymentConditionCode = "";
a.AuthCode = "ABC";
a.OptionalData =
"{"od":{"ver":"1.01","preauth":{"ver":"1.0","auth":{"ver":"1.0",
"token":{"ver":"1.0","t":"WZyq1XadTyso8AnpJdCT1GPKWUw=",
"e":"140903"}},
"data":{"ver":"1.0","id":4,"tpan":"521358*****2083",
"ref":"591504125853","TCC":"D@1","resp":"00"},
"receipt":{"ver":"1.0","cdt":{"ver":"1.0",
"prnstr":["MASTERCARD \",
"*****2083"]}}}}}}";
int result = Baxi.TransferAmount(a);
  
```





## 9.2 TopUp

These are typically used to load a mobile phone with more call time. See ref 1 for details.

The EAN number of the transaction is specified in the articleDetails parameter.

Buy a NetCom 150 TopUp code:

```

TransferAmountArgs a = new TransferAmountArgs();
a.OperID = "0000";
a.Type1 = 0x3B;
a.Amount1 = 1600;
a.Type2 = 0x30;
a.Amount2 = 0;
a.Type3 = 0x30;
a.Amount3 = 0;
a.HostData = "";
a.ArticleDetails = "07021940002810" + "\x1e"; // record separator
a.PaymentConditionCode = "";
a.AuthCode = "";
int result = Baxi.TransferAmount(a);
  
```

Receive the topup code from the OnTLDRReceived event, TLDRReceivedArgs contents:

RS = "\x1e", US = "\x1f"

TldType : "INFO"

TldData: "4005" + US + "00066" + US + "07021940002810" + US + "00000000833" + US + "7146417457411" + US + "0000004266-94-79" + US + "0015000" + RS

## 9.3 Gift Card

Gift card is implemented as a two transaction solution.

1. First do a normal purchase with the customer card.
2. Do a deposit with the same amount as the customer purchase.
3. Send the track 2 content of the giftcard with the TransferCardData() function.

The terminal will finish the deposit on the gift card.

## 9.4 Group ID

Group ID is a function used to separate cards within the same range based on values returned from the card.

This functionality is currently only supported by BBS cards.

This functionality is used in situations where you want to give different prices to customers with the same type of prepaid BBS cards, typically in canteens used by more than one company, where employees from the different companies gets different prices.

To retrieve the Group ID use the following sequence:

- Call the function SendTLD() specifying a Group ID request:

```
SendTldArgs a = new SendTldArgs();  
a.TldType = "REQ";  
a.TldField = "3000";  
Baxi.SendTLD(a);
```

- Get the group ID from the OnTLDReceived event, TLDReceivedArgs contents:

```
TldType : "RESP"  
TldData : "3000" + US + "0005" + US + "12345" + RS
```

## 9.5 Get encrypted card data

Call the function SendTLD() specifying a request encrypted card data:

```
SendTldArgs a = new SendTldArgs();  
a.TldType = "REQ";  
a.TldField = "3007";  
Baxi.SendTLD(a);
```

Read the encrypted card data in the OnTLDReceived event, TLDReceivedArgs contents:

```
TldType : "RESP"  
TldData : "3007" + US + "0015" + US + "0000000000000000" + RS
```

This currently only works in Norway.

## 9.6 Get encrypted card data and issuer ID

Call the function SendTLD() specifying a request encrypted card data and issuer:

```
SendTldArgs a = new SendTldArgs();  
a.TldType = "REQ";  
a.TldField = "3008";  
Baxi.SendTLD(a);
```

Read the encrypted card data in the OnTLDReceived response, TLDReceivedArgs contents:

```
TldType : "RESP"  
TldData : "3008" + US + "0015" + US + "0000000000000000" + RS
```

This currently only works in Norway.

## 9.7 Send a track 2 card swipe to the terminal:

```
TransferCardDataArgs a = new TransferCardDataArg();
a.TrackType = 0x32;
a.TrackData = "9571921000000000236";
Baxi.TransferCardData(a);
```

## 9.8 TIP

To be able to use tip, the service has to be enabled for the terminal.

This can be ordered from your dealer.

If the amount received in the amount field in the Local Mode message from the terminal is larger than the amount sent to the terminal in the Transfer Amount message, the difference should be treated as tip.

## 9.9 Check host connection

Call the function SendTLD() specifying a request for host connection status:

```
SendTldArgs a = new SendTldArgs(); a.TldType = "REQ"; a.TldField =
"1012"; Baxi.SendTLD(a);
```

Read the host connect status in the OnTLDRetrieved response, TLDReceivedArgs contents:

Ex connected to host:

```
TldType : "RESP"
TldData : "1012" + US + "0001" + US + "0" + RS
```

## 9.10 Set terminal language

Call the function SendTLD() specifying setting terminal language.

Ex English:

```
SendTldArgs a = new SendTldArgs();
a.TldType = "CMD";
a.TldField = "1014" + US + "0002" + US + "en" + RS;
Baxi.SendTLD(a);
```

## 9.11 Receive a card inserted status event

The terminal can be set to send events when a card is inserted or removed.

Read the status in the OnTLDRetrieved function.

Ex card inserted:

```
TldType : "STATUS"
TldData : "1013" + US + "0001" + US + "1" + RS
```

## 9.12 "Get Customer Info"

**WARNING:** This functionality is only supported on terminal versions 07.1X and higher. It must be turned off for older releases.

The generic TLD format is defined in [1].

The ECR invokes the Get Customer Info by sending a TLD request with the following tags in the given order:

```
"2000" + US + "0003" + US + "001" + RS +
"2002" + US + "0003" + US + "001" + RS +
"2001" + US + "0000" + US + RS
```

The tags *must* be in the order above for the request to work.

The ITU typically returns the customer information from the above request in the following manner:

```
"2000" + US + "0003" + US + "001" + RS +
"2002" + US + "0003" + US + "001" + RS +
"2001" + US + "0032" + US + "00987654 " + RS +
"2003" + US + "0002" + US + "00" + RS
```

When the ITU sends a TLD Status "Card inserted with Issuer ID" message, it typically contains the following (for a BankAxept card, issuer 30):

```
"1013" + US + "0001" + US + "1" + RS +
"3002" + US + "0010" + US + "0000000030" + RS
```

To receive the issuer ID tag in the card status message, the parameter for sending issuer ID with card status must be enabled in the ITU.

There are 2 normal use cases for the Customer Info functionality.

1. The ECR always initiates a GetCustomerInfo, before the purchase. (Automatic retrieval of Customer Info is turned on in the ITU). It is then the ITU which decides if there should be a request to the host, based on Issuer ID and terminal configuration.
2. The ECR receives "card inserted with issuer ID" from the ITU and based on issuer ID elects to initiate a customer info request or not.

Below are these 2 use cases described in detail. The data within the messages will typically be as described above.

#### 1. ECR initiates GetCustomerInfo.

The parameter for sending card status messages should be disabled for this scenario. This is because it is not needed by the ECR and it can also increase the risk of collisions and retransmissions.

Automatic customer info retrieval is enabled.

Sequence:

- The ECR will always initiate a transaction with a GetCustomerInfo request to the ITU through a TLD Request.
- The terminal goes into Bank Mode.
- The terminal will either ask for a card, or start processing if a card is already inserted or swiped.
- The terminal will return the Customer Info to ECR through a TLD Response and remains in bank mode. It is awaiting a TransferAmount transaction.
- The ECR processes the Customer Info and initiates a transaction. It is also possible to send an Admin Cancel (or Admin finish, see below) at this stage, if the ECR is not interested in any additional actions other than the customer info data.
- The ITU will complete any transaction or cancellation with a Local Mode message

#### 2. ECR reacts to Card Inserted.

Automatic customer info retrieval is disabled. "Card status messages with Issuer ID" is enabled in the ITU.

Sequence:

- The ITU will send a TLD Status message with "Card inserted and Issuer ID" to the ECR, when the customer inserts or swipes a card in the ITU.
- ECR processes the Issuer ID and decides if it should send a Get Customer Info request or not.
- ECR sends a Customer Info TLD Request.
- The ITU goes into Bank Mode.
- The ITU requests the customer info from the host and sends a TLD Response with Customer Info to the ECR.
- The ITU then awaits further action from the ECR and remains in Bank Mode.
- The ECR processes the Customer Info and normally sends a TransferAmount message to the ITU.
- The ITU will handle the transaction like normal and complete it with a Local Mode message.

## 9.13 "Get Customer Info" with Admin Finish.

For the "Get Customer Info" dialogues or any other TLD request dialogue, the terminal can remain in Bank Mode and expect a final transaction to take place.

There are occurrences where such a transaction will not take place and the ECR wishes to force the Terminal out of bank mode and back to idle.

This effectively completes the operation/dialogue and makes the terminal ready for a new operation/dialogue. This has been achieved with Admin Cancel in the past.

This has the unwanted behaviour of the terminal aborting the transaction in full, including display texts and showing "Cancelled by operator" in its display etc.

The Admin Finish command is an alternative to Admin Cancel in these cases, where it is better for the terminal to silently abort and revert to Local Mode and showing an idle screen.

## 9.14 Investigating results of last financial transaction.

If an error occurs in the middle of a transaction, BAXI will return an error and a local mode result of "Unknown".

If such an error or other errors make the result of a transaction unavailable or incomplete, one can use the administration functions "Print Latest Financial Transaction Receipt" and "Send Latest Financial Transaction Result" to fetch the result from the ITU.

This is a complement to the manual routines to provide an integrated solution for this problem for ECR vendors.

Let us assume that a transaction is unknown. To retrieve all necessary information, the ECR will have to perform the following:

```
int result = Baxi.Administration(new AdministrationArgs(0x313C, "0000"));
```

This results in a PrintText event, which will contain a copy receipt of the latest financial transaction result.

This is identical to entering the ITU menu and manually print out the copy receipt.

After the Local mode is received (administration is done), the ECR can send the next administration message to get the Latest Financial transaction result, which contains the Local mode data from the last financial transaction.

```
int result = Baxi.Administration(new AdministrationArgs(0x313D, "0000"));
```

This results in a LastFinancialTransactionResult event, which contains the local mode identical data from the last financial transaction.

The administration operation is then completed with a local mode message.

## 9.15 JSON replacing TLD.

### 9.15.1 CardInfo RequestAllTags(3999) using sendJson

As described in chapter 6 sendJson() replaces sendTld(). ECR can query Baxi to retrieve any information available about the current card.

If a card is not present then it will be requested by the terminal after this call.

### 9.15.2 sendTld

As reference this is how it is used with sendTld():

```
SendTldArgs a = new SendTldArgs();  
a.TldType = "REQ";  
a.TldField = "3999";  
Baxi.SendTLD(a);
```

OnTLDReceived returns the following where US is replaced by " " and RS by linefeed in the example:

```

3000 0000
3001 0016 <six digits>*****<four digits>
3002 0001 3
3003 0003 752
3004 0001 Y
3005 0001 Y
3009 0000
3011 0003 D__
3012 0008 STANDARD
3013 0000
3014 0000

```

### 9.15.3 sendJson

This is how it is done using sendJson() to send the same thing using the following JSON:

```

{
  "cardinfo": {
    "ver": "1.00",
    "alltags": "?"
  }
}

```

Code:

```

SendJsonArgs args = new SendJsonArgs(
    "{\n" +
    "  \"cardinfo\": {\n" +
    "    \"ver\": \"1.00\", \n" +
    "    \"alltags\": \"?\" \n" +
    "  }\n" +
    "}");
baxi.sendJson(args);

```

OnJsonReceived returns the following:

```

{
  "cardinfo": {
    "ver": "1.00",
    "iccgripid": "",
    "pan": "<six digits>*****<four digits>",
    "issid": "3",
    "countrycode": "752",
    "restrictions": "Y",
    "fee": "Y",
    "track2": "",
    "tcc": "D__",
    "bankagent": "STANDARD",
    "track3": "",
    "loyaltyinfo": ""
  }
}

```

## 9.16 JSON and Multiterminal

A single terminal can be configured with different TerminalId and therefore allow for separated configuration.

To use sendJson with Multiterminal add the optional tag "o" and "merch" with the six digit merchant number to the JSON call.

The Merchant Number is the first six digits of the TerminalId. This is also seen on the receipt as TERM: <6 digit MerchantNr><2 digit id>-<6 digit MerchantNr> i.e. TERM: 11223344-112233.

In this example the Merchant Number is 112233 and the TerminalId 11223344.

For single TerminalId terminals this optional "merch" tag is ignored.

```
"o": {  
  "ver": "1.00",  
  "merch": 112233  
}
```

Used with CardInfo RequestAllTags the JSON call will look like this:

```
{  
  "cardinfo": {  
    "ver": "1.00",  
    "alltags": "?"  
  },  
  "o": {  
    "ver": "1.00",  
    "merch": 112233  
  }  
}
```

## 9.17 JSON and VAS (Value Added Services)

Any number of Value Added Services can be used through the transparent Baxi JSON interface towards the Host.

JSON is sent using `sendJson()` in Baxi and the response comes back in `OnJsonReceived` in the form of another JSON.

Specific VAS will not be detailed here as this guide will remain service agnostic.

## 9.18 JSON and Terminal as an ECR Printer

The ECR print request/response messages are used for the feature of Terminal as an ECR printer that can handle text, barcode and bitmap print requests from ECR.

You can build the ECR print request in a JSON format(see **Appendix D** for the detailed information of JSON tags).

JSON is sent using `sendJson()` in Baxi and the response coming back in `OnJsonReceived` event in the form of another JSON.

If your ECR print request message size is large, Baxi will automatically split it into several ECR print requests and send them one by one.

The `onLocalMode` event will be fired as an consequence.

The result of `onLocalMode` will determine if the print process was successful or not.

Please ensure that JSON request doesn't contain any redundant spaces while using JSON tags.

## 10 MultiCurrency functionality

### 10.1 Introduction & Setup

MultiCurrency functionality is used to make card transactions using desired terminal profile selected by the OperID.

The correct OperID and currency code are settled by Baxi.NET by using the offline BIN -tables(s).

There can be up to 10 different profiles in terminal for different currencies.

- Baxi.NET gets the PAN from terminal as a result for CardInfo Json
- Baxi.NET iterates through offline BIN table and adds corresponding "baxi\_currencycode" and "baxi\_operid": to Json Repsonse
- ECR uses baxi\_currencycode to set the correct amount for TransferAmount
- ECR uses "baxi\_operid" in TransferAmount to detect desired terminal profile

#### 1. Enable multicurrency in baxi.ini

[MULTICURRENCY]

# Set when using multicurrency functionality(Several terminal profiles with different currencies)

# MultiCurrencyPath defines the path where MultiCurrency is located.

# If let blank default path is the location of baxi\_dotnet.dll

# Set path with backslash at the end eg. c:\baxi\

UseMultiCurrency=1

MultiCurrencyPath=""

#### 2. Edit MultiCurrency.xml

MultiCurrency.Xml have to be located in same directory as baxi\_dotnet.dll

<!-- This is BAXI.NET Multicurrency configuration file -->

<!-- OperID identify the logical terminal ID that shall be used in the transaction. Legal values 0001-0010. 0001 is the master TID. -->

<!-- CurrencyCode defines the currency used with OperID -->

<!-- CurrencyFile defines the file which icludes the PANs mapped to OperID -->

<!-- All PANs which are not in CurrencyFiles fall into OperID without CurrencyFile -->

```
<?xml version="1.0" encoding="utf-8" ?><MultiCurrency>
<Terminal OperID="0001" CurrencyCode="978" />
<Terminal OperID="0002" CurrencyCode="752" CurrencyFile="Currency752.txt" />
<!-- <Terminal OperID="0003" CurrencyCode=" " CurrencyFile=" " /> -->
<!-- <Terminal OperID="0004" CurrencyCode=" " CurrencyFile=" " /> -->
<!-- <Terminal OperID="0005" CurrencyCode=" " CurrencyFile=" " /> -->
<!-- <Terminal OperID="0006" CurrencyCode=" " CurrencyFile=" " /> -->
<!-- <Terminal OperID="0007" CurrencyCode=" " CurrencyFile=" " /> -->
<!-- <Terminal OperID="0008" CurrencyCode=" " CurrencyFile=" " /> -->
<!-- <Terminal OperID="0009" CurrencyCode=" " CurrencyFile=" " /> -->
</Multicurrency>
```

### 10.2 Testing multicurrency

1. Send Json cardinfo with one of the terminals merchant ID



```
{
  "cardinfo": {
    "ver": "1.00",
    "alltags": "?"
  },
  "o": {
    "ver": "1.00",
    "merch": 733300
  }
}
```

2. Insert card and select card type if required
3. Check that baxi\_currencycode and baxi\_operid exist in Json response and are according to setup in MultiCurrency.xml  
Json:

```
{
  "cardinfo": {
    "ver": "1.00",
    "iccgripid": "",
    "pan": "527592*****4796",
    "issid": "4",
    "countrycode": "246",
    "restrictions": "-",
    "fee": "-",
    "track2": "",
    "tcc": "I__",
    "bankagent": "",
    "track3": "",
    "loyaltyinfo": "",
    "baxi_currencycode": "752",
    "baxi_operid": "0002"
  }
}
```

4. Make TA with OperID returned in JSON -response and check that correct terminal profile is used for transaction

## 11 Best practice implementation of security in the ECR interface.

These are recommendations regarding some of the security functions offered with Baxi and Nets terminals.

Recommended practice is to implement all functions with high security.

These recommendations are based on experienced skimming practices which mainly have been:

1. Skimming equipment is mounted on the terminal when the shop is closed.
2. The shops terminal is replaced with a terminal with skimming equipment mounted.

Powercyclecheck is designed to counter scenario 1, while the other functions are designed to counter scenario 2.

Function	Low security	Medium security	High security
<b>VendorInfo extended.</b>  Data reported by ECR to terminal. Readable by Nets.	Identification of pos application supplier in the vendorinfoextended string	Identification of pos application supplier and Identification of site in the vendorinfoextended string	Identification of pos application supplier and unique Identification of POS in the vendorinfoextended string
<b>Powercyclecheck</b>  Checks when the terminal has been tuned on and off.	Turned off	Turned on	Action demanded from the POS operator on the OnDisplayText generated
<b>Tidcheck</b>  Checks if the TID (Terminal ID) of the terminal has changed.	Turned off	Turned on	Turned off in Baxi.ini, but implemented tid check in the POS application with action required from the pos operator on tid change, Preferable with operator authentication.
<b>Serial number check</b>  Checks if the physical serial number of the terminal has changed.	No serial number check.	Implement serial number check in the POS application with action required from the pos operator on serial number.	Implement serial number check in the POS application with action required from the pos operator on serial number change with operator authentication.

## 12    **Deprecated Functions**

These functions will be supported for backwards compatibility.

## 13 Removed Functions

These functions have been removed entirely and are no longer supported

### 13.1 TransferAmount\_V2

Replaced by TransferAmount\_V4.

Same as TransferAmount\_V4 but without the parameter PaymentCardCondition and AuthCode.

### 13.2 TransferAmount\_V3

Replaced by TransferAmount\_V4.

Same as TransferAmount\_V4 but without the parameter AuthCode.

### 13.3 TransferAmount\_V4

### 13.4 Administration( int admCode, string OperId)

Replaced by the new Args type interface Administration(AdministrationArgs args).

### 13.5 SendTLD( string typeString, byte[] tldField)

Replaced by the new Args type interface SendTLD(SendTldArgs args).

### 13.6 TransferCardData( int trackType, string trackData)

Replaced by the new Args type interface TransferCardData(TransferCardDataArgs args).

### 13.7 BiBAdministration( int AdmCode, string OperID)

Replaced by the new Args type interface BiBAdministration( BiBAdministrationArgs args).

### 13.8 BiBTransaction( int amount, string transactionData)

Replaced by the new Args type interface BiBTransaction( BiBTransactionArgs args).

## 14 Appendices

### 14.1 Appendix A – Status codes

#### 14.1.1 Error codes

Error code	Description
2011	Error when sending message
2012	Error when receiving message
6006	Baxi locked
7001	Unknown Error
<b>Error code</b>	<b>MsgRouter errors</b>
2100	MsgRouter Error Range starts with 2100, this code is not used.
2101	General msgrouter error
2102	Msgrouter timeout
2103	Socket error
2104	Message length error
2105	Sending of message failed
2106	Connection error
	<b>Socket (Server) Errors (Range 2111-2199)</b>
2111	General socket error
2112	Socket timeout
2113	Socket error
2114	Message length error
2115	Message sending failed
2116	Socket connection error
7005	Open rejected
7007	Obsolete terminal version
7008	No host contact.
7009	No response from terminal
7010	Create log directory fail
7011	Open IO fail
7012	Unexpected frame
7013	Close rejected
7015	Unknown terminal frame
7016	Terminal Reboot Detected

#### 14.1.2 Method reject codes

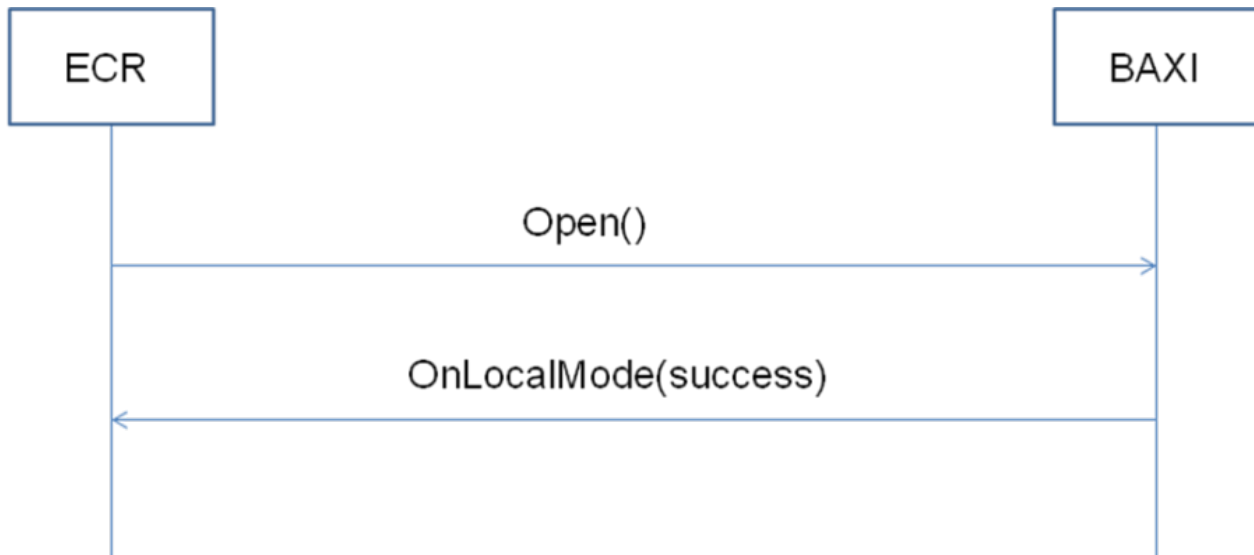
Method Reject code	Description
--------------------	-------------

	<b>General cases</b>
7100	Processing previous command
7101	Unable to process
7102	Already Open
7103	Not Active
7104	Terminal busy, administration
	<b>Property bad values</b>
7401	Log file prefix
7402	Log file path
7403	Host IP address
7404	Vendor info extended
7405	Trace level
7406	Baud rate
7407	COM port
7408	Host port
7409	Indicate EOT transactions
7410	Cutter support
7411	Printer width
7412	Display width
7413	Power cycle check
7414	TID supervision
7415	Auto get customer info
7416	DeviceString
7420	Terminal ready
7421	UseDisplayTextID
7422	UseExtendedLocalMode
7423	MsgRouterIpAddress
7424	MsgRouterOn
7425	MsgRouterPort
7426	UseSplitDisplayText
7427	DisplayTextInLM
7428	Always Use Total Amount In ExtendedLM
7429	Client ID
7430	Socket Listener
7431	Bluetooth Tunnel
	<b>Function argument bad values</b>
7501	Only track 2 support

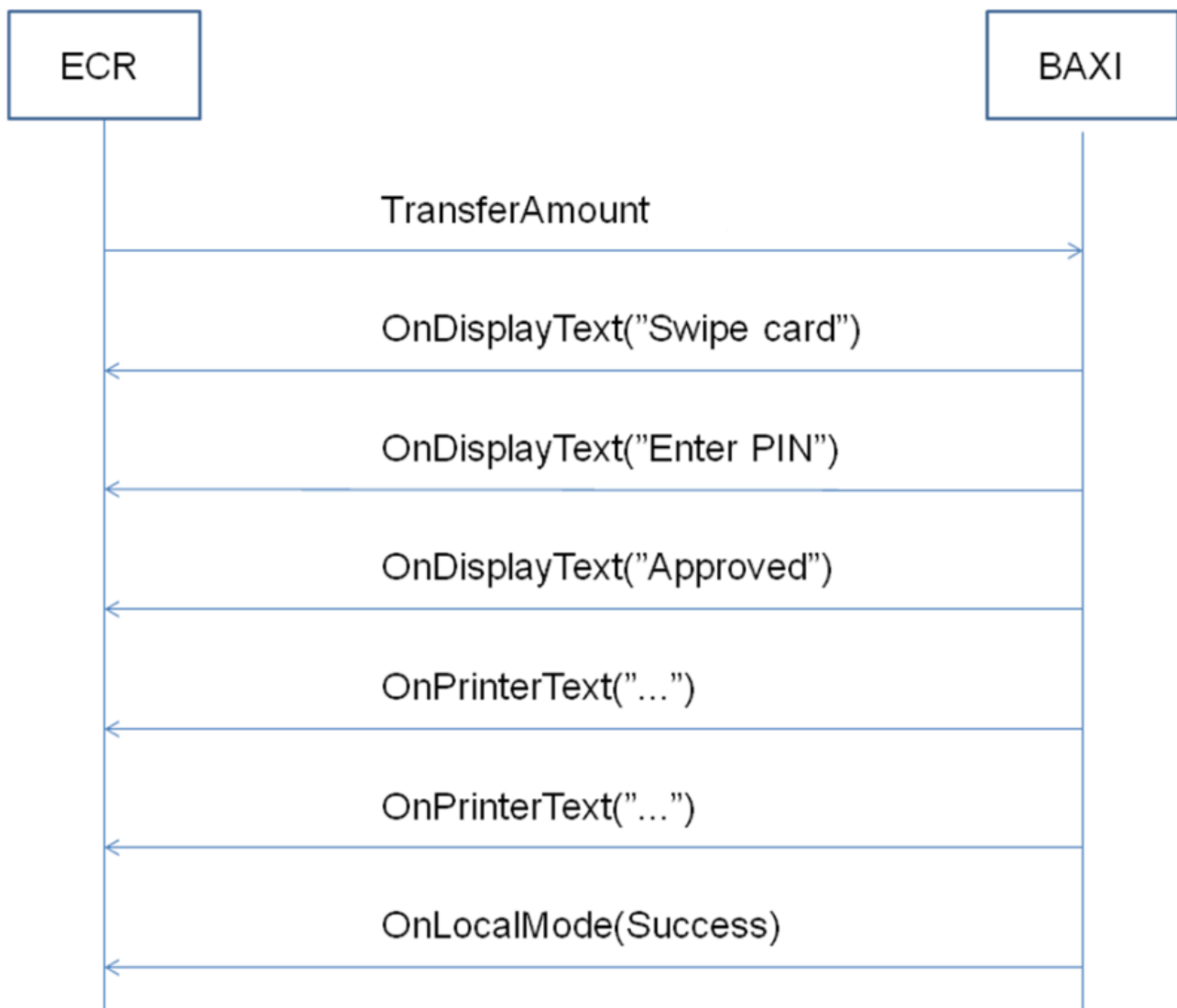
7502	Invalid track length
7503	Transfer amount invalid type
7504	Transfer amount data too long
7505	Transfer amount article details too long
7506	Invalid operator ID
7507	Invalid administration code
7508	TLD unknown type
7509	TLD bad field value
7510	TLD could not build
7511	Transfer amount PCC too long
7512	Transfer amount PCC not alphanumeric
7513	Transfer amount AuthCode too long.
7514	Transfer amount AuthCode not alphanumeric.
7515	JSON Bad field value
7516	Transfer amount Optional Data too long.
	<b>Multi instance cases</b>
7601	Multi instance clientid not existing
7602	Multi instance clientid in use
7603	Multi instance bad config file
	<b>Multi currency cases</b>
7700	PAN not numeric
7701	Load failed
7702	Operator ID missing
7703	File error
7704	XML error
7705	Currency code missing

## 14.2 Appendix B – Sequence diagrams

### 14.2.1 Initialisation

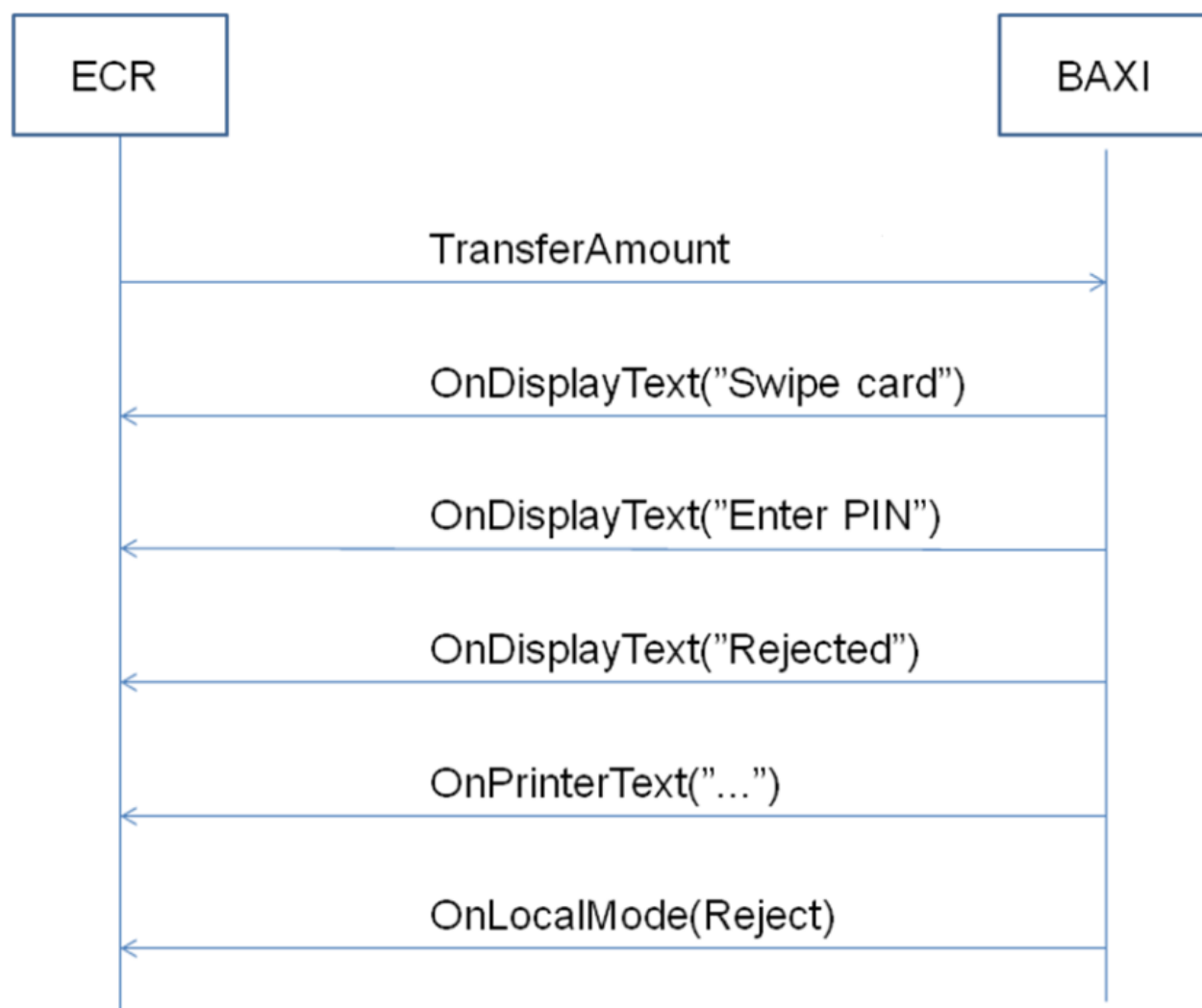


#### 14.2.2 Successful purchase

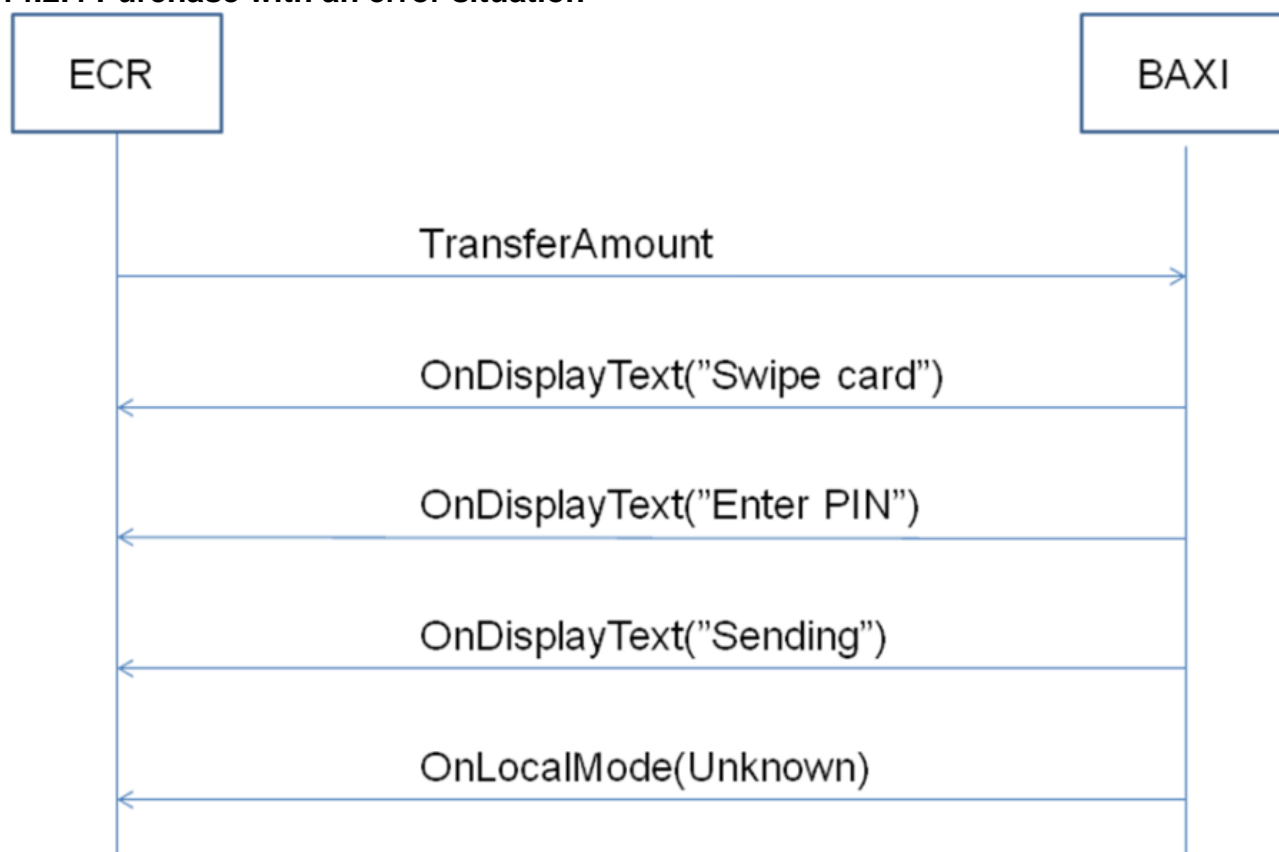


#### 14.2.3 Unsuccessful purchase





#### 14.2.4 Purchase with an error situation



## 14.3 Appendix C – ECR Print Request\Response

Message Type	Tag Name	JSON Tag	Object type	Mandatory\ Optional	Length (bytes)	Description
Request/Response	Print message	printmsg	Object	M	-	Current version: 1.00
Request	Print lines	rows	Array	M	-	
Request	Print content	type	String	M	-	Possible values: "txt": Text content "bcode": Barcode "bmp": Bitmap
<b>Text content related JSON tags</b>						
Request	Print Text	data	String	O	-	For ex. "data": "sample text" Prints a line with text as "sample text". "data": "*" Prints a line filled with the given char. "data": " " Prints a blank line. Note: The print width depends on font and text width given. Refer table 4.4.1 below.
Request	Text font	font	String	O	-	Possible values: "normal" (default value) "bold" "small" "large"
Request	Text width	w	String	O	-	Possible values: "fix" (default value) "prop"
Request	Text alignment	align	String	O	-	Possible values: "left" (default value) "center" "right"
Request	Reverse	reverse	Integer	O	-	Possible values: 0 (default value) 1
Request	Blank lines	blank	Integer	O	-	Possible values: <= 20 Note: Add this tag to add 15 lines in the end of receipt to get receipt out of printer.
<b>Barcode content related JSON tags</b>						

Request	Barcode symbol	symbol	String	M	-	Possible values: "code128" "code25" "code39" "ean8" "ean13" "pdf417" "qrcode"
Request	Barcode data	data	String	M	-	
<b>1D barcode symbol specific tags</b>						
Request	Barcode 1 bar height	h	Integer	O	-	Possible values: >= 50 pixels Default as 50 pixels
Request	Barcode 1 bar width	w	Integer	O	-	Default as 2 pixels
Request	Direction	dir	Integer	O	-	Possible values: 0: Horizontal (default value) 1: Vertical
Request	Alignment	align	String	O	-	Possible values: "left" (default value) "center" "right"
Request	Print barcode text	value	Integer	O	-	Possible values: 0: Don't print (default value) 1: Print
<b>2D barcode symbol specific tags</b>						
Request	Rotate barcode	rotate	Integer	O	-	Possible values: in degrees 0 (default value) 90 180 270 Note: tag applies to PDF417 only.
Request	Barcode size	size	Integer	O	-	Allowed range: Between 63 and 384, both values inclusive. Default values are used depending on barcode text. Note: tag applies to qrcode only.
Request	Barcode mode	mode	Integer	O	-	Possible values: in degrees 0: Numeric mode (default value) 1: alphanumeric mode 2: 8 bit data mode 3: kanji (shift-jis) mode Note: tag applies to qrcode only.
<b>Bitmap content related JSON tags</b>						
Request	Bitmap data	data	String	M	-	Data must be Base64 encoded.
Request	X coordinate	x	Integer	O	-	Allowed range: in Pixels [0,383]

Request	Y coordinate	y	Integer	O	-	Allowed range: [0, bitmap height]
Request	Reverse	reverse	Integer	O	-	Possible values: 0 (default value) 1

Terminal automatically moves the remaining text into next line if the length of the string received, exceeds than the number of characters that can be printed in a single line based on font and text width.

Font	Text Width	Allowed number of characters in a line (Excluding null char)
normal	fix	24
	prop	24 - 35
bold	fix	11
	prop	14 - 21
small	fix	48
	prop	52 - 65
large	fix	12
	prop	14 - 17

ECR print response possible status texts description:

Text print errors	Description
-1	The font is not in the font file, nothing is printed
-2	Character in the message is not in the font.
-3	Margins are too big, nothing is printed
PDF417 barcode errors	Description
-1	Columns must be between 1 and 30, inclusive.
-2	Rows must be between 3 and 90, inclusive.
-3	Error Correction level should be from 1 to 8.
-4	Size of barcode too large.
-5	Data will not fit in 90 rows. Must reduce input data or use more columns.
-6	Must have more than two columns.
-7	Message length plus Error Code Length too big. Use larger row or column size. Or set rows = 999 for unlimited row size
-8	Incorrect rotate
-9	Size of barcode too large
-10	Size of barcode too large
QR barcode errors	Description
-1	Size must be between 63 and 384, inclusive.
-2	Failed to encode barcode text.
-3	Error Correction level should be from 0 to 4.
-4	Encode mode should be from 0 to 4.
-5	Data will not fit in 90 rows. Must reduce input data or use more columns.

-6	Must have more than two columns.
-7	Message length plus Error Code Length too big. Use larger row or column size. Or set rows = 999 for unlimited row size
-8	Incorrect rotate
-9	Size of barcode too large
-10	Failed to encode barcode text
-11	QR Decoder DLL not present
<b>Bitmap errors</b>	<b>Description</b>
-1	BMP Bad drivers
-2	Given BMP is too big.
-3	BMP wrong video : bad bmp format
-4	BMP compression failure : bad bmp format
-5	No volume found
-6	No file found
-7	Not enough memory
-8	Read error
-9	BMP not available

For example:

```
{ "printmsg": { "ver": "1.0", "rows": [ { "type": "bmp", "data": "Qk0+BwAAAAAAD4AAAAoAAAAzwAA
AEAAAAABAAEAAAAAAAAHADEdGAAxA4AAAAAAAAAAAAAAAAAAP///wD////////////////////////////////////
////////////////////////////////////gAA////////////////////////////////////4AAP////////////////////////////////////+AA
D///A////////////////////////////////////gAA///wAAf////////////////////////////////////4AAP/wAAA/
////////////////////////////////////+AAD/wAAAD////////////////////////////////////gAA/wAAAAAP////////
////////////////////////////////////4AAP/4AAAAAB////////////////////////////////////+AAD/4AAAAAP////////////////////////////////
////////////////////////////////gAA/8AAAAAA////////////////////////////////4AAP+AAAAAAH////////////////////////////////
/+AAD/AAAAAAB///AP//Af///gAD//AAH8AA//gAA/gAAAAAAP///wB//gD//wAAA/+AAA/AAAP4AAP4AA
AAAAAB///8Af/4A//gAAAP/AAAPwAAA+AAD8AAAAAAP///AH/+AP/wAAAD/gAAD8AADgAA/AAAAAAB///
wB//gD/4AAAA/4AAA/AAAA4AAPgAAAAABwf//8Af/4A/8AAAAAP8AAPwAAAAGAADwAAAAAP/z///AH/+AP+A
AAD/AAAD8AAABgAA8AAAAAP9///wB//gD/gAAAA/wAAA/AAAAIAAOAAAAAHAA//8Af/4A/wAf///4AP/
///ACAADgAAAAHgAH//AH/+AP8AP///+AH/////4AgAA4AAADgAAf//wB//gD/AH///gD/////AIAA
MAAABwAAH//8Af/4A/wB///4A/////wCAADAAAAA4AAA//AH/+AP4A///+AP///8AgAA4AAAACAA
Af//wB//gD+AP///gD///+AIAAOAAAAOAAAH//8Af/4A/gD///4A/////gCAADwAAADAAAB//AH/+
AP4A///+AP///wAgAA8AAABgHwA//wB//gD+AP///gD/////4AIAAPgAAA8P/wf//8Af/4A/gD///
4A/////wACAAD4AAAf/g/P///AH/+AP4A+AAD+AP///AAAgAA/AAAP/AB///wB//gD+APgAA/gD///+AA
AYAAP4AAH/AAH//8Af/4A/gD4AAP4A///+AAAOAAD/AAH/gAAf///AH/+AP4A+AAD+AP///AAAHgAA/+AH
/wAAD///wB//gD+APgAA/gD//gAAH4AAPj+P/wAAA//8Af/4A/gD4AAP4A///4AAP+AADwP/w8AAH///
AH/+AP4A//gD+AP//8AA//gAA8APA0AAAA//wB//gD+AP/8A/gD//AB//4AAOAAAHAAAAAP//8Af/4A/gD
//AP4A//wB//+AADgAADgAAD//AH/+AP4A//wD+AP//8Af//gAAwAAAwAAAAf//wB//gD+AP/8A/gD//
/AH//4AAMAAAYAAAAH//8Af/4A/wD/+AP4A//wB//+AADAAAMAAAAB//AH/+AP8Af/gD+AP//8Af//gAA
4AAOAAAAAf//wB//gD/AD/4A/gD//AH//4AAPAAHAAAAAP//8Af/wA/wAf8Af4A//wB//+AAD8APgAAAA
D///AHvwAP+AA4AH+AHf38AD/7gAA//gAAAAA//wAAAAH/gAAD/gAAB/AAAA4AAP/gAAAAAf//8AAAA
B/8AAAA/4AAAf4AAAOAAD4AAAAAAH//AAAAAf/gAAf+AAAH+AADgAA+AAAAAAD//wAAAAAP/8AAP/
gAAB/wAAA4AAPwAAAAAAB//8AAAAAD//gAAH/4AAAf+AAAOAAD+AAAAAAAf//AAAAAD//+AAH/+AAAH/4AA
DgAA/gAAAAAAP//wAAAD///+Af//gD///4AA4AAP8AAAAAAH////////////////////////////////4A////////+AAD/gAA
AAAD////////////////////////////////+AP////////gAA/8AAAAAB////////////////////////////////gD////////4AAP/gAAAAA////////
////////////////////////////////4A////////+AAD/+AAAAAf////////////////////////////////+AP////////gAA//wAAAAf////////
////gD////////4AAP//AAAAf////////////////////////////////4A////////+AAD/8AAAf////////////////////////////////+AP//
////gAA//4AAf////////////////////////////////gD////////4AAP///P////////////////////////////////w////////+AAD
////////////////////////////////////////gAA", "x": 100}, { "type": "txt", "data": "NETS BRANCH
NORWAY", "align": "center"}, { "type": "txt", "data": "Haavard Martinsensvei
54", "align": "center"}, { "type": "txt", "data": "TLF:22 21 60
47", "align": "center"}, { "type": "txt", "blank": 1}, { "type": "txt", "data": "NETS BRANCH
NORWAY", "align": "center"}, { "type": "txt", "data": "05.08.15
16:45
KASSE:001", "font": "small"}, { "type": "txt", "data": "ReceiptNo:123456789", "font": "small
"}, { "type": "txt", "data": "-"}, { "type": "txt", "data": "Item
No.1
5000,00", "font": "small"}, { "type": "txt", "data": "Item
No.2
999,00", "font": "small"}, { "type": "txt", "data": "-"}, { "type": "txt", "data": "Total
5999,00"}, { "type": "txt", "data": "
-----"}, { "type": "txt", "blank": 2}, { "type": "txt", "data": "Thanks for the
visit!!", "align": "center"}, { "type": "bcode", "data": "123456789", "symbol": "code25", "al
ign": "center"}, { "type": "txt", "blank": 14} ] ] }
```

Receipt output:



NETS BRANCH NORWAY

Haavard Martinsensvei 54

TLF: 22 21 60 47

06.02.15 16:45

KASSE:001

Receipt No:123456789

-----  
Item No.1 5900,00Item No.2 999,00  
-----Total 5999,00  
-----

Thanks for the visit !!

