



EFT/POS ECR-Interface

# MPD (Multi Protocol Driver) Manual

## Technical Specification

Version  
Date  
State  
Security Note  
Document No.

V1.14  
14.12.2017  
Released  
Public  
MPDMANV114009EN

**© 2017 SIX Payment Services Ltd.**

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: Dezember 2017

# Table of Contents

<b>Part I Introduction</b>	<b>1</b>
<b>Part II Change History</b>	<b>1</b>
<b>Part III Function Overview</b>	<b>2</b>
1 Operations Overview	2
2 Transaction Operation	2
3 Transaction Programming Flow	4
4 Open, Close, Balance and Submit Operations	5
5 AutoMode Overview	5
6 DCC Transaction Operation	5
DCC Transaction flow: all ok	6
DCC Transaction flow: not ok for cashier	6
DCC Transaction flow: not ok on host	7
7 topup Transactions (mobile voucher)	7
GetTopupIssuers	8
8 GiftCard Transactions	8
9 Coupon Transactions	8
10 pay@the Table	9
pay@theTable Terminal requests	9
pay@theTable ECR answers	9
pay@theTable flow	9
pay@theTable shift open method additional parameters	9
11 Multi Terminal and multi ECR operation	9
Multi Terminal and VEZ+overIP	9
Multi ECR operation	10
Multi MPD operation	10
12 Petrol	10
Display Text	10
Display Resource	11
Get Key Input	11
Petrol Card PIN Check	12
Petrol Eject Card	13
Petrol Key Code	13
Petrol Transaction Flows	13
Petrol Sale Item	15
13 Loyalty Transaction Operation	15
Loyalty Transaction flow: with Loyalty Information	16
Loyalty Transaction flow: without Loyalty Information	16
<b>Part IV Runtime Library</b>	<b>17</b>
1 Runtime Library	17
2 Error Codes and Messages	17
3 Timeouts	18

<b>4 Driver Object</b>	<b>19</b>
<b>Driver Object</b>	<b>19</b>
<b>Methods</b>	<b>20</b>
.....Open Method.....	20
.....Close Method.....	21
.....Transaction Method.....	21
.....Transaction Options.....	23
.....Commit Method.....	23
.....CommitPartial Method.....	24
.....Balance Method.....	24
.....Validation Method.....	24
.....Abort Method.....	25
.....QueryStatus Method.....	25
.....DeviceControl Method.....	25
.....DeviceAccess Method.....	26
.....IccControl.....	26
.....Complete Method.....	28
.....Connect Method.....	28
.....Disconnect Method.....	28
.....ReadProperty Method.....	28
.....WriteProperty Method.....	29
.....PrintOnEFT Method.....	29
PrintText	29
Print Options	30
.....CommitAmount Method.....	30
<b>Properties</b>	<b>31</b>
.....ActivationState Property.....	31
.....ApplicationKey Property.....	31
.....Applications Property.....	31
.....Async Property.....	31
.....BrandId Property.....	31
.....CardExpiration Property.....	31
.....Cashier Property.....	32
.....CompletedDeviceEvent Property.....	32
.....CompletedCommand Property.....	32
.....DccMode Properties.....	32
.....DeviceEventCode Property.....	33
.....DeviceStatusCode Property.....	33
.....DriverAddress Property.....	33
.....ECR Information Properties.....	34
.....ExceptionCode, ExceptionMessage Properties.....	34
.....Language Property.....	34
.....MPD Version Properties.....	35
.....NativeStatus, NativeMessage Properties.....	35
.....OperationMode Property.....	35
.....Receipt Properties.....	35
.....RefNumber Property.....	36
.....Status Property.....	37
.....Time Property.....	37
.....Topup Properties.....	37
.....Track2 Property.....	38
.....AutoMode Property.....	38
.....PrintCharSet Property.....	39
.....PrinterState Property.....	39
.....TipAmount Property.....	39
.....CustomerLanguage Property.....	40
.....DeviceId Property.....	40

.....pay@theTable Properties .....	40
.....NbrlInstalments Property .....	40
.....SuppressReceiptOnTerminal Property .....	40
.....Petrol Properties .....	40
.....OperationPhase Property .....	41
.....AmountOther Property .....	41
.....FeeAmount Property .....	41
.....AcqID Property .....	41
.....StatKeyPANRctInd Property .....	41
.....SubBrand Property .....	42
.....Application Label Property .....	42
.....AccountIndex Property .....	42
.....ECRSeqCounter Property .....	42
.....LoyaltyMode Properties .....	42
.....LoyaltyData Property .....	43
.....LoyaltyInfo Property .....	43
<b>Events</b> .....	<b>43</b>
.....OnStatusChange Event .....	43
.....OnCommandComplete Event .....	43
.....OnError Event .....	43
.....OnDeviceEvent Event .....	43
.....OnDeviceMsg Event .....	44
<b>5 Applications Collection</b> .....	<b>44</b>
Applications Collection .....	44
<b>6 Application Object</b> .....	<b>45</b>
Application Object .....	45
Application Type .....	45
<b>7 Receipt Object</b> .....	<b>45</b>
Receipt Object .....	45
<b>8 Devices Collection</b> .....	<b>46</b>
Devices Collection .....	46
<b>9 Device Object</b> .....	<b>46</b>
Device Object .....	46
<b>Part V Integration</b> .....	<b>47</b>
<b>1 Integration under Windows</b> .....	<b>47</b>
Windows NT Systems (NT4, 2000, XP, 2003, Vista, 2008, 7 and Windows 8 .....	47
.....Using under Windows NT Systems .....	47
.....Visual Basic Script Example .....	47
Windows CE .....	47
.....The Windows CE Binaries .....	47
<b>2 Integration under Linux</b> .....	<b>47</b>
Integration under Linux .....	47
<b>3 Integration under OSX</b> .....	<b>48</b>
Integration under OSX .....	48
<b>Part VI Using COM</b> .....	<b>48</b>
<b>1 Using the COM/OLE Interface</b> .....	<b>48</b>
<b>Part VII Using Java</b> .....	<b>48</b>
<b>1 Using the Java interface</b> .....	<b>48</b>
<b>2 Using the Events in Java</b> .....	<b>49</b>

3	Java ECR Simulator	49
<b>Part VIII</b>	<b>Using .NET</b>	<b>50</b>
1	Using the .NET interface	50
<b>Part IX</b>	<b>Using EFTAPI</b>	<b>51</b>
1	EFTAPI overview	51
2	EFTAPI Applications	52
<b>Part X</b>	<b>Configuration and Debugging</b>	<b>52</b>
1	Configuration and Debugging	52
2	Configuration with XML	55
3	Work with the EFTSimulator	57
<b>Part XI</b>	<b>Trace and Debug Options</b>	<b>57</b>
<b>Part XII</b>	<b>Socket Level Access</b>	<b>58</b>
1	Socket Connection	58
2	EFT Control Protocol	59
3	driver-definition.xml	59
4	connectproperties-request Message	59
5	open-request Message	60
6	open-response Message	60
7	close-request Message	61
8	close-response Message	61
9	transaction-start Message	61
10	transaction-auth Message	62
11	transaction-finish Message	63
12	transaction-end Message	63
13	abort Message	64
14	balance-request Message	64
15	balance-response Message	64
16	query-status Message	65
17	device-control-request Message	65
18	device-control-response Message	65
19	exception Message	66
20	status Message	66
21	application-info Message	66
	<b>Index</b>	<b>68</b>

# 1 Introduction

## MPD (Multi Protocol Driver) Manual

### Introduction

This is the documentation to the version V1.14

# 2 Change History

## Change History

The following table lists the changes and added features with each major release

*version*

<b>1.00</b>		initial version
<b>1.02</b>	introduced	DCC (direct currency conversion) O.P.I. - mode configuration with XML possibility of more than one device on MPD
<b>1.04</b>	introduced	DeviceAccess method, transaction type TopUp, property MPDVersion, DCC one phase mode functionality of the davinci VENDING
<b>1.06</b>	introduced	topup mobile voucher, VEZ+overIP , KESS Gateway
<b>1.08</b>	introduced	CXI - mode and c-credit Terminal support ExtraXML mode (e.g. RPT) new receipt handling Giftcard Mobile Couponing pay@theTable (pay@theTable is the technical and ComfortPay commercial name) new ECR Information properties
<b>1.10</b>	introduced	eps42 protocol for vending machines between ECR (Handelskasse) and MPD print request method negative receipt (for each transaction request a receipt will be generated) different shift open requests for pay@theTable (with or without user password etc.) new Property TipAmount new property CustomerLanguage
<b>1.12</b>	introduced	Petrol (SIX IFSF Terminal) support UPI - mode (Unified Payment Interface) new transaction type "payment with cashback" new transaction type "reservation cancellation" real card data, which the mpd receives from the cash register will be answered with the error 618 (invalid track data) so only technical cards will be accepted. The goal is that the mpd is out of the scope of PCI. The log storage time can be defined using a startup parameter
<b>1.14</b>	introduced	SIXml "POS ECR" Protocol support new Properties StaticKeyPanIndex, AcqId, ApplicationLabel, SubBrand, AccountIndex and ECRSeqCounter new Option for partial approved transaction new Transaction types for Gift- and Loyalty- Cards new commit method "commitamount" introduced new device control class "9" introduced new petrol transaction flow, with SalesItem, Restrictioncodes Loyalty change pay@theTable not supported anymore

## 3 Function Overview

### 3.1 Operations Overview

#### Operations Overview

A set of basic operations is supported by all EFT devices. Additionally functions may be supported for particular EFT devices or EFT/ECR protocol types.

#### Financial Operations

- The [Open](#) and [Close](#) operations are used to change the EFT shift state. They are carried out when the operator is logged on, resp. off the ECR. This operation is supported by all devices. The Open operation must be performed before transactions can be carried out.
- The [Transaction](#) operation is used for all types of financial operation carried out on the device. A set of subtypes of this operation exists to support use cases such as reservations, cancellation and tipping. This operation is supported by all devices, but some functions may not be implemented.
- The [Balance](#) operation requests per-contract counter totals from the device and triggers an end-of-day procedure. All devices support this operation. It should be executed at least once a day to trigger the financial flow, except for EFT devices that are configured for automatic balance operation.

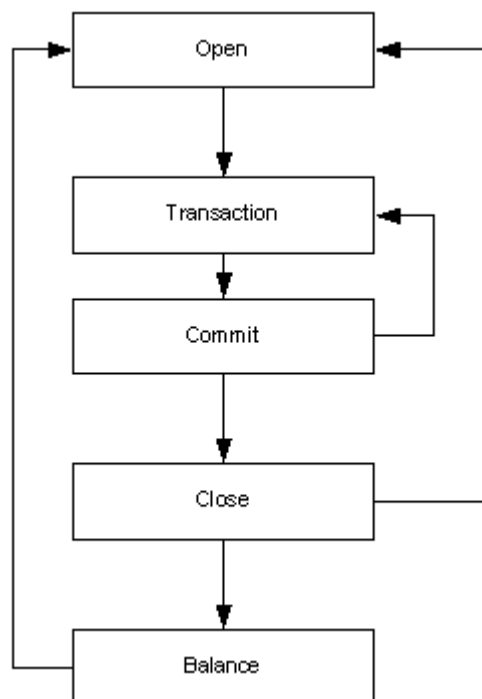
The result of a financial operations contains receipt data that shall be printed by the ECR application.

#### Administrative Operations

- Status information may be sent by the EFT device whenever some event of interest for the ECR occurs.
- The [DriverControl](#) Operation can be used by the ECR to trigger a home-call/initialization process on the EFT device or other functions, depending on the EFT device.

#### Operation Sequence

The following illustrates a possible sequence of financial operations on the EFT device. Similar diagrams will be used to describe the sequence of particular transaction types.



Note that the Transaction operation consists of two steps: Transaction and Commit are separated for consistency reasons.

### 3.2 Transaction Operation

#### Transaction Operation

##### See Also

[Transaction Method](#)  
[Commit Method](#)



### Summary

This operation is used to perform financial transfer activities on the EFT device. This involves debit and credit transactions, reservations, reservation payments and reversals. The type of transaction is determined the transaction function-code. The commit method is used to tell the EFT whether to commit or to rollback the transaction once it has successfully been authorized.

### Debit and Credit Transactions

This set of functions involves all types that are not related to any previous transaction and that cause financial flow. No reference-number is required for these functions.

### Reservation Transactions

The reservation function does not cause financial flow; it is a preauthorization. Only debit reservations are possible.

### Reserved Transactions

The reference-number of the reservation must be supplied ( [TrxRefNum](#) property) .

### Reversal Transactions

The reference-number of the reversed transaction must be supplied ([RefNumber](#) property). Only the immediately preceding transaction may be reversed.

### Referral Transactions

A referral transaction requires the [AuthCode](#) to be set.

### DCC Transactions

see [DCC Transaction operation](#).

### topup Transactions (mobile voucher)

see [topup Transactions](#).

### GiftCard Transactions

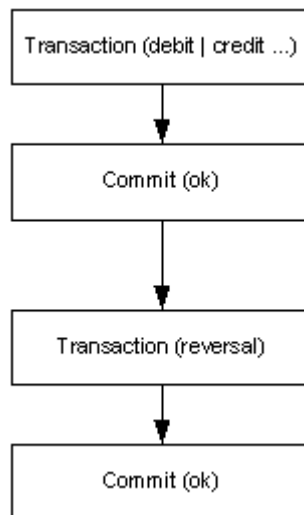
see [GiftCard Transactions](#)

### Coupon Transactions

see [Coupon Transactions](#)

### Petrol Transactions

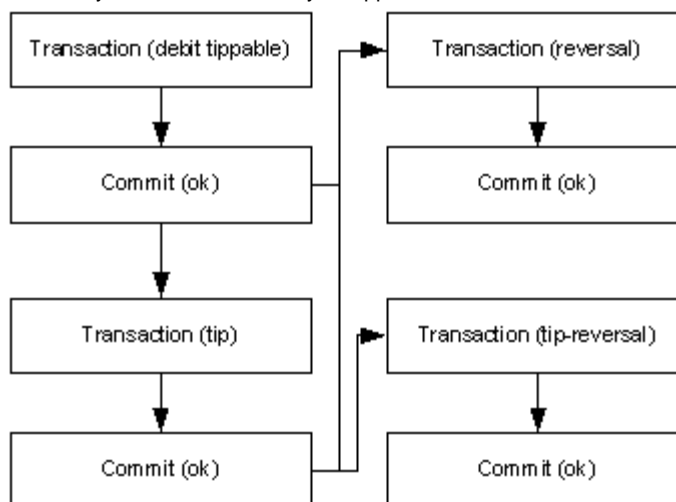
see [Petrol Transaction Flows](#)



The diagram illustrates the operations for a payment-reversal sequence.

#### Tip Transactions

The reference-number of the tipped transaction must be supplied ([RefNumber](#) property). Tip amount may be reversed by use of the tip-reversal function. Only debit-transaction may be tipped.



#### Consistency Management

The transaction operation is always performed in two steps:

1. The operation is prepared and authorized by the EFT device. Receipt and authorization data is passed to the application for printing.
2. The payment transaction is either committed or aborted, depending on an application parameter.

#### Arguments

The following aspects of a transaction may be influenced by the application:

- The function type is determined by the function-code.
- The transaction amount
- An optional reference-number to reference a previous transaction
- Optional ECR identifier, Cashier-ID and Terminal-ID. The values may be ignored by the device.
- Choice of operation mode: POS, MOTO, e-commerce
- Optional Track-2 and Track-3 data

## 3.3 Transaction Programming Flow

#### Transaction Programming Flow

A *Transaction* is a timeduring process. During this process the *ReaderStatus* will change and the application may need to *Abort* the *Transaction*. Therefore there exist several possibilities to run the transaction:

- run the Transaction in **synchron** mode (see [Async Property](#))
  - change of ReaderStatus are reported with the [OnDeviceEvent Event](#)
  - the Transaction cannot be Aborted from the application
- run the Transaction in a separate **thread** in **synchron** mode (see [Async Property](#))
  - change of ReaderStatus are reported with the [OnDeviceEvent Event](#)
  - the Transaction can be Aborted from the main thread.
- run the Transaction in **asynchron** mode (see [Async Property](#))
  - change of ReaderStatus are reported with the [OnDeviceEvent Event](#)
  - the Transaction can be Aborted from the application
  - wait for completion by:
    - waiting for the [OnCommandComplete Event](#)
    - calling the [Complete Method](#) timed until completion

## 3.4 Open, Close, Balance and Submit Operations

### Open, Close and Balance Operations

See also

[Open Method](#)  
[Close Method](#)  
[Balance Method](#)  
[Receipt Property](#)

### Summary

Used to control terminal shift state and financial flow on the EFT device. The receipt data resulting from these operations may be printed to paper or saved in a file.

## 3.5 AutoMode Overview

### AutoMode Overview

With AutoMode, the automatic behaviour of the driver can be controlled. AutoMode queries periodically the status of the device and reports changes of [DeviceEventCode](#), [DeviceStatusCode](#) and [ActivationState](#) with the [OnDeviceEvent-Event](#).

AutoMode sets also the unique transaction behaviour for the different devices.

AutoMode can be set:

- as commandline parameter ( `/AutoMode` )
- AutoMode Property. This has priority over the commandline value.

See also [AutoMode Property](#)

## 3.6 DCC Transaction Operation

### DCC Transaction Operation

a DCC Transaction makes a direct currency conversion. (Customer pays amount in his foreign currency, merchant gets amount in his local currency. conversion is made by SIX).

this option is only possible on ep2 terminals, when DCC option is configured on this terminal.

This property is used to handle the DCC transactions (direct currency conversion).

The transaction is done in two phases:

- phase 1: get DCC information (foreign currency, amount in foreign currency, exchange rate)
- phase 2: book DCC transaction.

The transaction can also be done in one phase, then the DCC transaction is booked directly.

both phases are processed by a transaction method call, the flow is controlled by the [DccMode Property](#)

For more information about DCC information see [DccMode Properties](#)

DCC transaction flows:

[DCC Transaction flow: all ok](#)  
[DCC Transaction flow: not ok for cashier](#)  
[DCC Transaction flow: not ok on host](#)

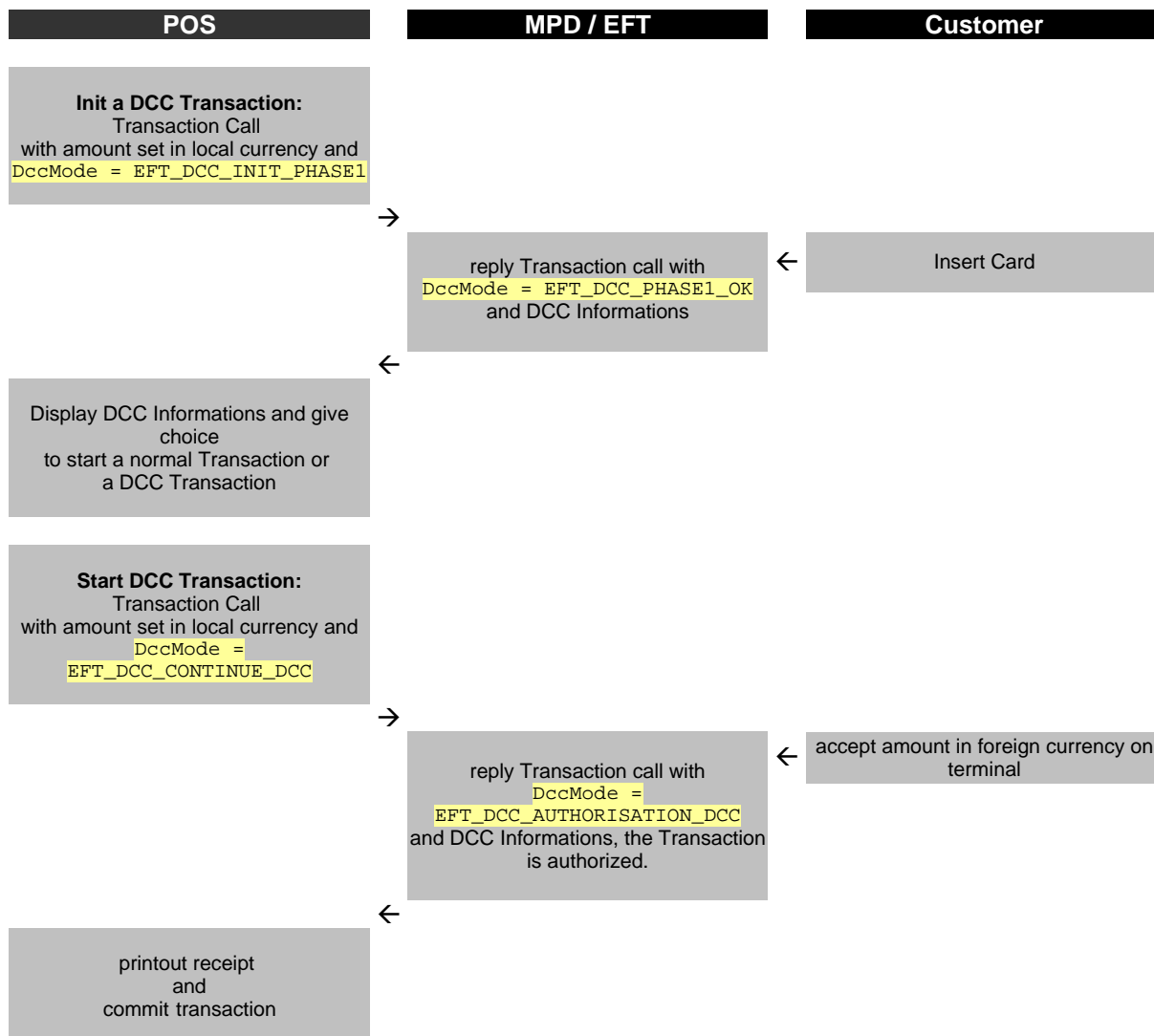
### Remarks

- the transaction can be aborted before a card is inserted and also in PHASE\_1\_OK state with `eft.Abort()`
- errors during the transaction are reported in the same way, as in a normal transaction.
- the driver is in the EFT\_S\_PREPARED state during the PHASE\_1\_OK state

- in the Applications Object, the Balanced DCC operations are reported with the currency "DCC"
- the DCC exchange rates table can be printed with [DeviceControl](#) 0x05 0x01

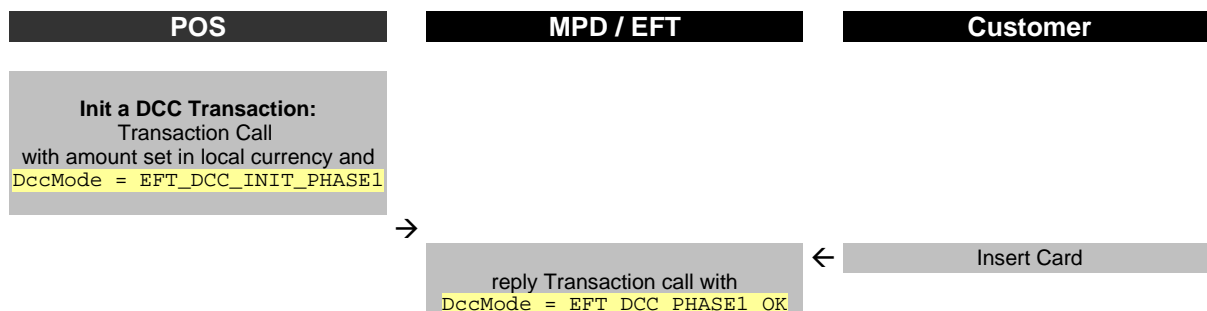
### 3.6.1 DCC Transaction flow: all ok

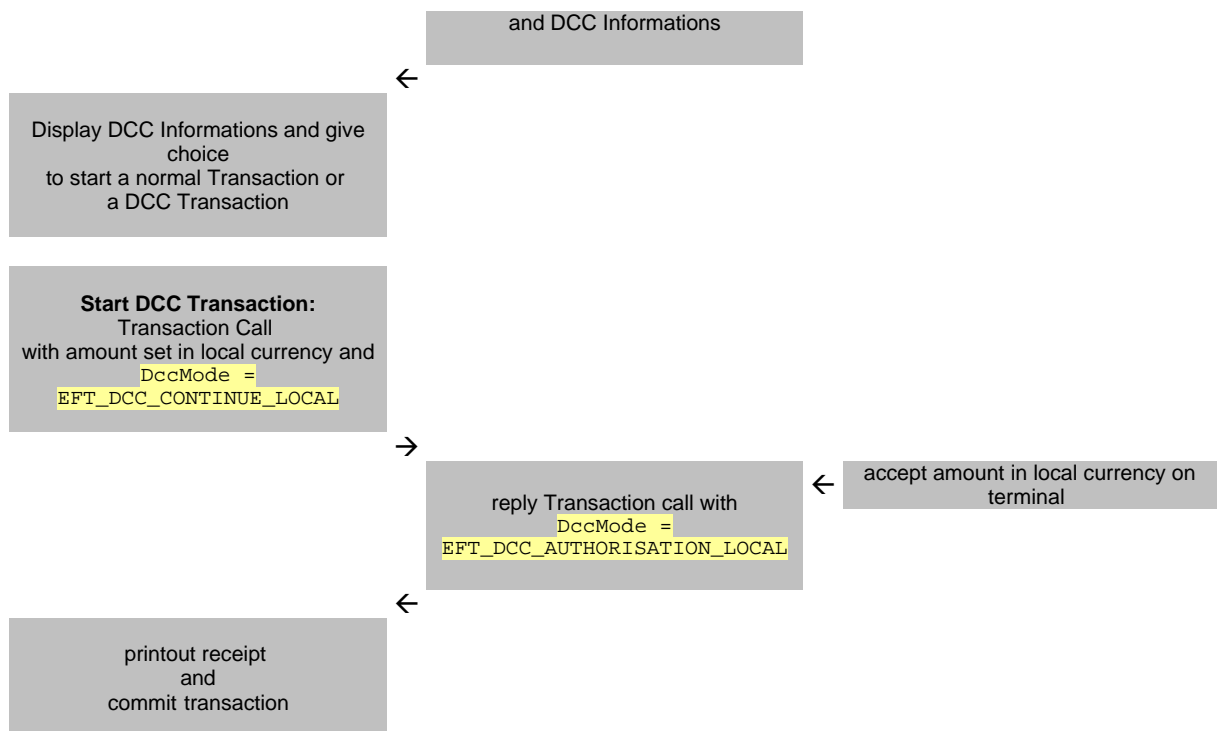
flow of a DCC Transaction with DCC is ok on host, DCC is ok for cashier and DCC is ok for customer:



### 3.6.2 DCC Transaction flow: not ok for cashier

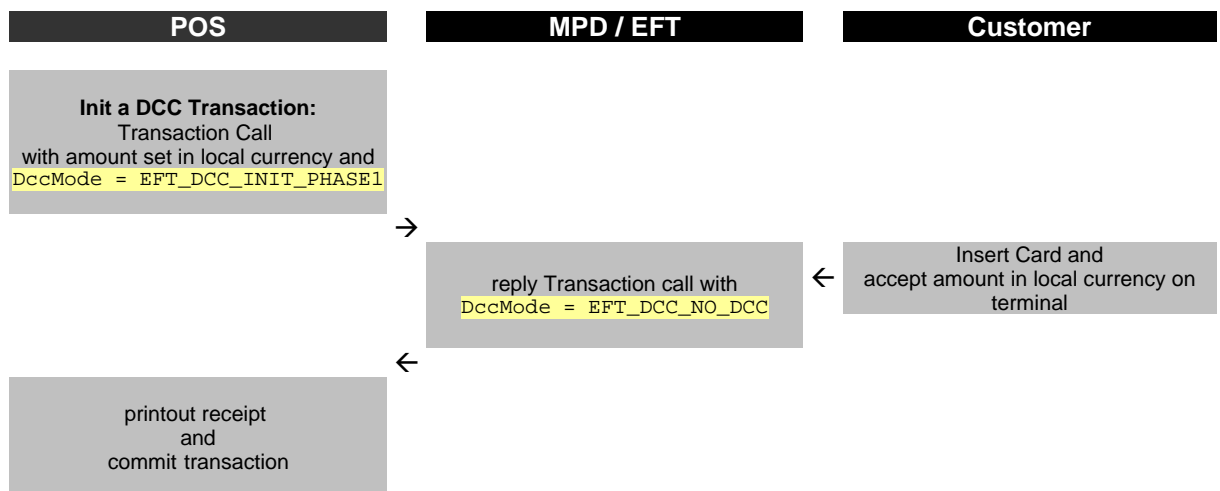
flow of a DCC Transaction with DCC is ok on host, DCC is not ok for cashier





### 3.6.3 DCC Transaction flow: not ok on host

flow of a DCC Transaction with DCC is not ok on host or terminal is not configured for DCC



## 3.7 topup Transactions (mobile voucher)

### topup Transactions (mobile voucher)

a topup transaction is used for mobile voucher (prepay refill on mobile phones).

for a topup transaction the transaction *FunctionCode* has to be set to **"topup"**.  
The property [TopupIssuerName](#) has to be set also, and a allowed amount has to be used.

Possible issuers and allowed amounts can be asked with [GetTopupIssuers](#).

As result of a topup transaction the customer receipt with the refill code and refill instructions is available in [ReceiptText](#), and the merchant receipt with transaction information is available in [ReceiptMerchantText](#).

### 3.7.1 GetTopupIssuers

#### GetTopupIssuers

gets the Topup issuer and the possible amount to be refilled.  
to ask, use DeviceControl 6, 1

```
eft.DeviceControl ( 6, 1 )
```

as result in the property **DeviceControlData** is:

TopupIssuerName1, possibleAmount1,possibleAmount2,...;TopupIssuerName2,...;  
eg:

```
Swisscom mobile,10,30,50,100;Sunrise,10,50,100;
```

## 3.8 GiftCard Transactions

#### GiftCard Transactions

GiftCard transactions are used to handle Gift Cards.

special Transaction Types are:

card-activation	Fill a new balance on a Gift Card
card-activation-reversal	Reversal of activation
load	load an amount to a Gift Card
load-reversal	Reversal of load
buy	buy points
buy-reversal	Reversal of buy points
collect	collect points
collect-reversal	Reversal of collect points
card-balance	Return the balance of the card (in property Saldo)

normal Transaction Types are:

debit	Payment with a Gift Card
reversal	Reversal of the debit Transaction

Gift Card transaction return with the Authorization the Balance of the Card:

Property:

SaldoFlag	(boolean) the Authorization includes a balance value, returned in "Saldo"
Saldo	(integer) Balance on GiftCard

## 3.9 Coupon Transactions

#### Coupon Transactions

Coupon transactions are used to handle Coupons.

special Transaction Types are:

coupon	coupon transaction
coupon-reversal	Reversal of a previous coupon transaction

Coupon number:

if the coupon number is entered at the ECR, this number has to be set in the property `TrxRefNum`

Reversal reference:

for `coupon-reversal` set the property `RefNumber` to the value of the transaction to be reversed

Coupon transaction returns with the Authorization the following properties:

Property:

CouponValueUnit	coupon value unit
CouponValue	value of coupon according to type
CouponType	type of coupon: 00: coupon for goods 01: coupon for fix amount

CouponPromNumber	02: coupon for percentage of purchase
CouponReferenceNumber	promotion number
RefNumber	transaction reference number
	transaction reference number (same as CouponReferenceNumber)

Coupon Reversal transaction returns with the Authorization the following properties:

Property:

CouponReferenceNumber	reference number of the reversed coupon transaction
-----------------------	---

### 3.10 pay@the Table

**pay@theTable (ComfortPay)**

not supported anymore

#### 3.10.1 pay@theTable Terminal requests

**pay@theTable Terminal requests**

not supported anymore

#### 3.10.2 pay@theTable ECR answers

**pay@theTable ECR answers**

not supported anymore

#### 3.10.3 pay@theTable flow

**pay@theTable flow**

not supported anymore

#### 3.10.4 pay@theTable shift open method additional parameters

**pay@theTable shift open method additional parameters**

not supported anymore

### 3.11 Multi Terminal and multi ECR operation

**multi Terminal and multi ECR operation**

More than one Terminal can be connected on a MPD. It can be connected on serial interface or with VEZ+overIP. see [multi Terminal and VEZ+overIP](#).

More than one ECR can use the same Terminal. see [multi ECR operation](#)

#### 3.11.1 Multi Terminal and VEZ+overIP

**multi Terminal and VEZ+overIP**

More than one Terminal can be connected on a MPD. It can be connected on serial interface or with VEZ+overIP. a ECR specifies the desired Terminal with the DeviceId.

On serial Interfaces, the DeviceId has to be defined in the [Configuration file](#).

On Devices, connected with VEZ+overIP, the Terminal-ID is used as DeviceId, or the DeviceId selects a TerminalId, defined in the [Configuration file](#).

the following scheme is used by the MPD to select the Terminal:

the definition in the config.xml is evaluated

if not defined, for DeviceId 0 and 1, the first Terminal is used.

if not 0 or 1, the Terminal with TID = DeviceId is used.

If the desired Terminal is not available, a broadcast message is sent into the LAN, to force the Terminal with the desired TID to connect to the MPD. If Device 0 or 1 is searched, all Terminals in the LAN are forced to connect. (if the configuration switch / TID is set, the Terminal with this TID is searched).

### 3.11.2 Multi ECR operation

#### multi ECR operation

One Terminal can be used by several ECRs.

one ECR can open the shift on the Terminal, it has to set the useMode Parameter to 1 (shared use) before the open command to make the Terminal available to be used by other ECRs. Only this ECR can close the shift on this Terminal.

#### GetDevice:

The Terminal can be connected to a ECR with the **GetDevice Method** ([DeviceControl](#) 0x20 0x04). the result can be *connected*, *used by other ECR* or *not available*. the state of the Terminal is signaled by a DeviceEvent with the appropriate [DeviceStatusCode](#).

If the Terminal is used by other ECR or not available at this time, the MPD waits for this Terminal until the ReleaseDevice Method is called. When the Terminal becomes available, it is connected and a DeviceEvent is fired with the appropriate [DeviceStatusCode](#).

#### ReleaseDevice:

The Terminal can be released with the **ReleaseDevice Method** ([DeviceControl](#) 0x20 0x05). A connected Terminal is released and is then available for other ECRs.

#### Operation on the Master ECR:

- set the DeviceId for the Terminal.
- set useMode to 1 (shared use)
- call GetDevice (this is not mandatory)
- open the Terminal (other operations are possible like close, balance etc)
- call ReleaseDevice

#### Operation on any ECR:

- set the DeviceId for the Terminal.
- call GetDevice
- make a **Transaction**
- call ReleaseDevice

### 3.11.3 Multi MPD operation

#### Multi MPD operation

More than one MPD can be started on a computer. then the switches [EFTPort](#) and [ECRPort](#) have to be set. also the Terminals have to be configured appropriately.

## 3.12 Petrol

#### Petrol

for the employment for gas stations the following functionality is available:

use of DeviceControl:

- Display Text
- Activating predefined Screens
- Key Input
- Petrol Card PIN Check

### 3.12.1 Display Text

#### Display Text

The application uses the DeviceControl Method for sending a free definable text to the EFT to be shown on the display (refer to [DeviceControl Method](#))

```
eft.DeviceControl (EFT_DCC_IFSF, EFT_DCA_IFSFDisplayText) or eft.DeviceControl (0x08, 0x01)
```

#### Properties in

IFSFText	string	Information text to display eg. "Welcome". see IFSFText
----------	--------	---

#### Properties out



none

#### Error Conditions

The DeviceControlResult 0 indicates the successfully executed command (see also [DeviceControl Method](#))

### 3.12.2 Display Resource

#### Display Resource

The application uses the DeviceControl Method for sending a free definable text to the EFT to be shown on the display (refer to [DeviceControl Method](#))

```
eft.DeviceControl (EFT_DCC_IFSF, EFT_DCA_IFSFDisplayResource) or eft.DeviceControl (0x08, 0x02)
```

#### Properties in

IFSFResource	integer		Information text to display eg. Resource 60 "please enter your card". see IFSFResource
IFSFTimeout	integer	optional	time to display the resource text. see IFSFTimeout

#### Properties out

none

#### Error Conditions

The DeviceControlResult 0 indicates the successfully executed command (see also [DeviceControl Method](#))

### 3.12.3 Get Key Input

#### Get Key Input

The application uses the DeviceControl Method for initiating the PIN checking procedure (refer to [DeviceControl Method](#))

```
eft.DeviceControl (EFT_DCC_IFSF, EFT_DCA_IFSFGetKeyInput) or eft.DeviceControl (0x08, 0x03)
```

#### Properties in

IFSFTimeout	integer	Cardholder Input timeout for entering the key. see IFSFTimeout
IFSFResource	integer	Resource ID refer to IFSF Resources
IFSFInputCharsMin	integer	minimum number of characters
IFSFInputCharsMax Amount	integer	maximum number of characters

#### Properties out

DeviceControlData

#### Error Conditions

The DeviceControlResult 0 indicates that the entered key is available in DeviceControlData property and command was successfully (see also [DeviceControl Method](#))

#### IFSF Resources

- 2 bill total: {Amount} pump selection: [\_\_\_\_] and OK
- 5 pump selection: [\_\_\_\_] and OK
- 7 mileage: [\_\_\_\_] and OK
- 8 carnumber: [\_\_\_\_] and OK
- 9 drivercode: [\_\_\_\_] and OK
- 10 further info: [\_\_\_\_] and OK
- 48 please enter your drivercode: [\_\_\_\_]
- 50 Enter Fleet ID:
- 57 enter vehicle or driver code: [\_\_\_\_]
- 58 bill total: {Amount} choose a pomp
- 60 please enter your card
- 98 please select a pump: [\_\_\_\_] and OK
- 105 bills and cards are available:
- 107 cards available bills out-of-order:

```

120 bill total: {Amount} car washing area: [____] and OK
121 bill total: {Amount} car washing program: [____] and OK
122 bill total: {Amount} product: [____] and OK
123 select car washing area: [____] and OK
124 select car washing program: [____] and OK
126 select product: [____] and OK
150 load card amount: [____] and OK
156 pump selection: [____] and OK/or card
157 car washing area: [____] and OK/or card
158 car washing program: [____] and OK/or card
159 product: [____] and OK/or card
186 bill total: {Amount} continue with OK
244 amount: {Amount} ok/stop
400 Product {Product} {Amount} Preis neu

```

### 3.12.4 Petrol Card PIN Check

#### Petrol PIN Check

The application uses the DeviceControl Method for initiating the PIN checking procedure (refer to [DeviceControl Method](#))

```
eft.DeviceControl (EFT_DCC_IFSF, EFT_DCA_IFSFPinCheck) or eft.DeviceControl (0x08, 0x04)
```

#### Properties in

IFSFTimeout	integer		Cardholder Input timeout for entering the PIN. see IFSFTimeout
IFSFText	string		Information text to display eg. "PLEASE ENTER YOUR PIN". see IFSFText
IFSContextId	integer	mandatory	has to be taken from CardInfos
IFSInputCharsMin	integer		minimum number of characters
IFSInputCharsMax	integer		maximum number of characters
IFSLastTry	integer		0

#### Properties out

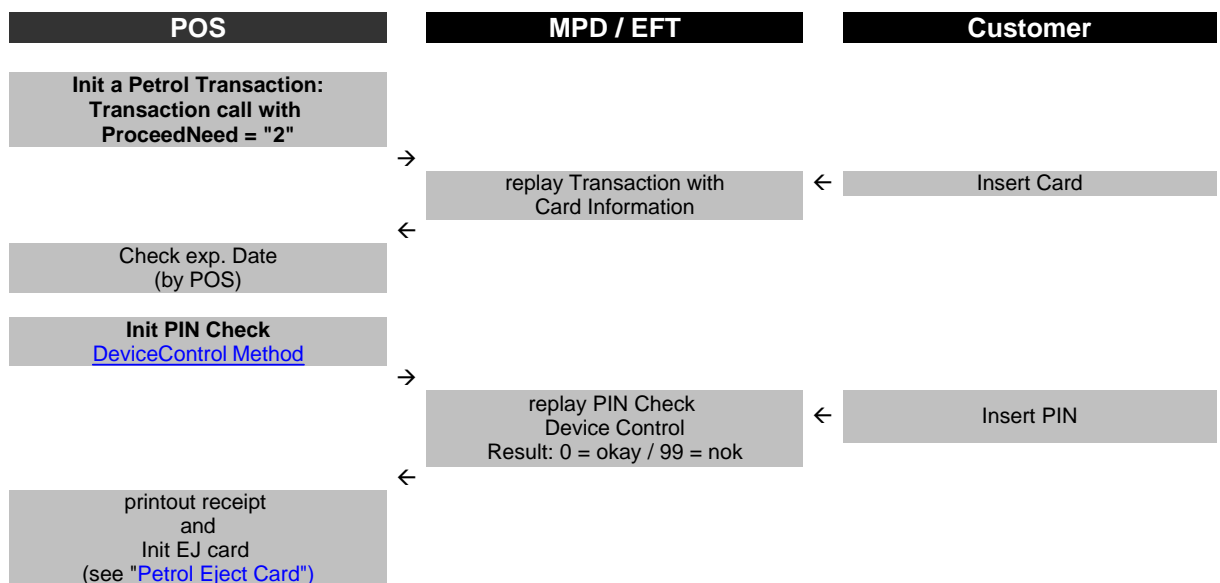
none

#### Error Conditions

The DeviceControlResult 0 indicates that the PIN verification was successfully (see also [DeviceControl Method](#))

#### Indoor Flow PIN Check (MOC offline)

flow of indoor PIN Check with petrol transaction start:



The pin check can also started when card data are available.

### 3.12.5 Petrol Eject Card

#### Petrol Eject Card

The application uses the DeviceControl Method for ejecting the card (refer to [DeviceControl Method](#))

The application can make use of the **SuccessIndicator** property that can be specified together with the Eject Card request to control the appearance of the success/error icon on the cardholder display.

```
eft.DeviceControl (EFT_DCC_CARD_READER, EFT_DCA_CR_CARD_EJECT)
```

#### Properties in

IFSFText	string	Information text to display eg. "Use Pump #1". see IFSFText
IFSFSuccessIndicator	integer	5 = Display with OK Sign 6 = Display with NOK Sign

#### Properties out

none

#### Error Conditions

see [DeviceControl Method](#)

### 3.12.6 Petrol Key Code

#### Petrol Key Code

An event indicates that a key was pressed on the cardholder keypad (refer to [OnDeviceMsg Event](#))

#### Function Keys

Function keys are numbered from left to right, top to bottom using key codes 0x81, 0x82, ... respectively.

#### OK, CORR, STOP, MENU, POINT Key

OK	0x0D
CORR	0x1B
STOP	0x18
MENU	0x12
POINT	0x2E

#### Numeric Keys

0..9	0x60
------	------

### 3.12.7 Petrol Transaction Flows

#### Petrol Transaction Flow

The Petrol solution supports Indoor and Outdoor terminals. The use cases and their respective flows are different between these types of terminals.

Transaction Types are:

#### preauthorization debit-preaut

Preauthorization of an amount  
Make debit transaction based on previous preauthorization (financial advice).  
The [RefNumber](#) property and the Amount must be set to the referenced preauthorization.

This property is used to handle the special petrol transaction see also [Petrol Properties](#)

#### Properties in

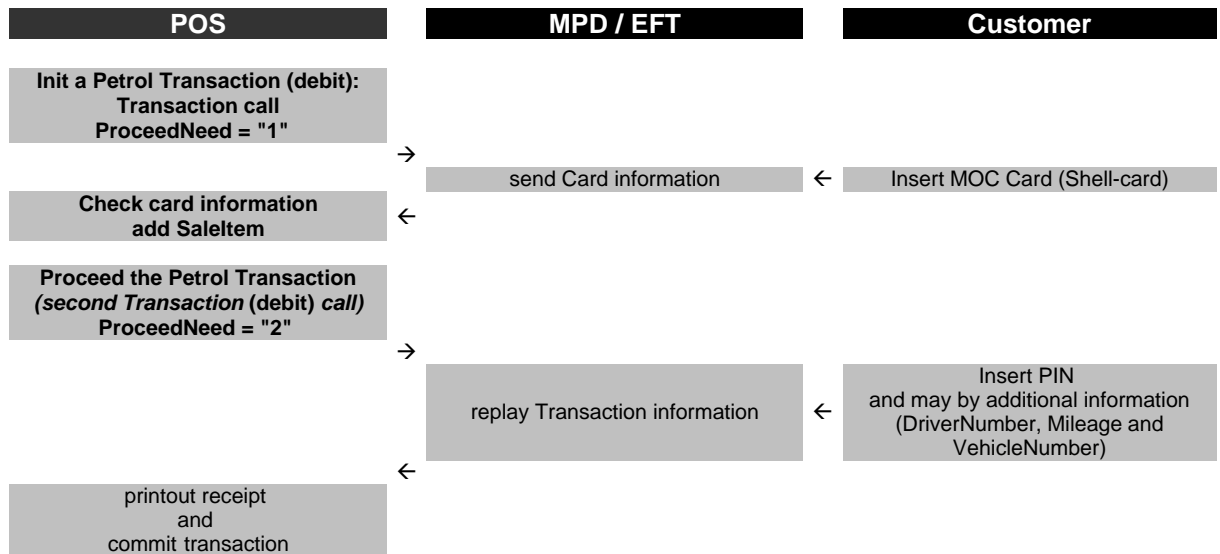
ProceedNeeded	integer	The use of ProceedNeeded option is limited to the transaction start before card information are available.
---------------	---------	--

#### Remarks

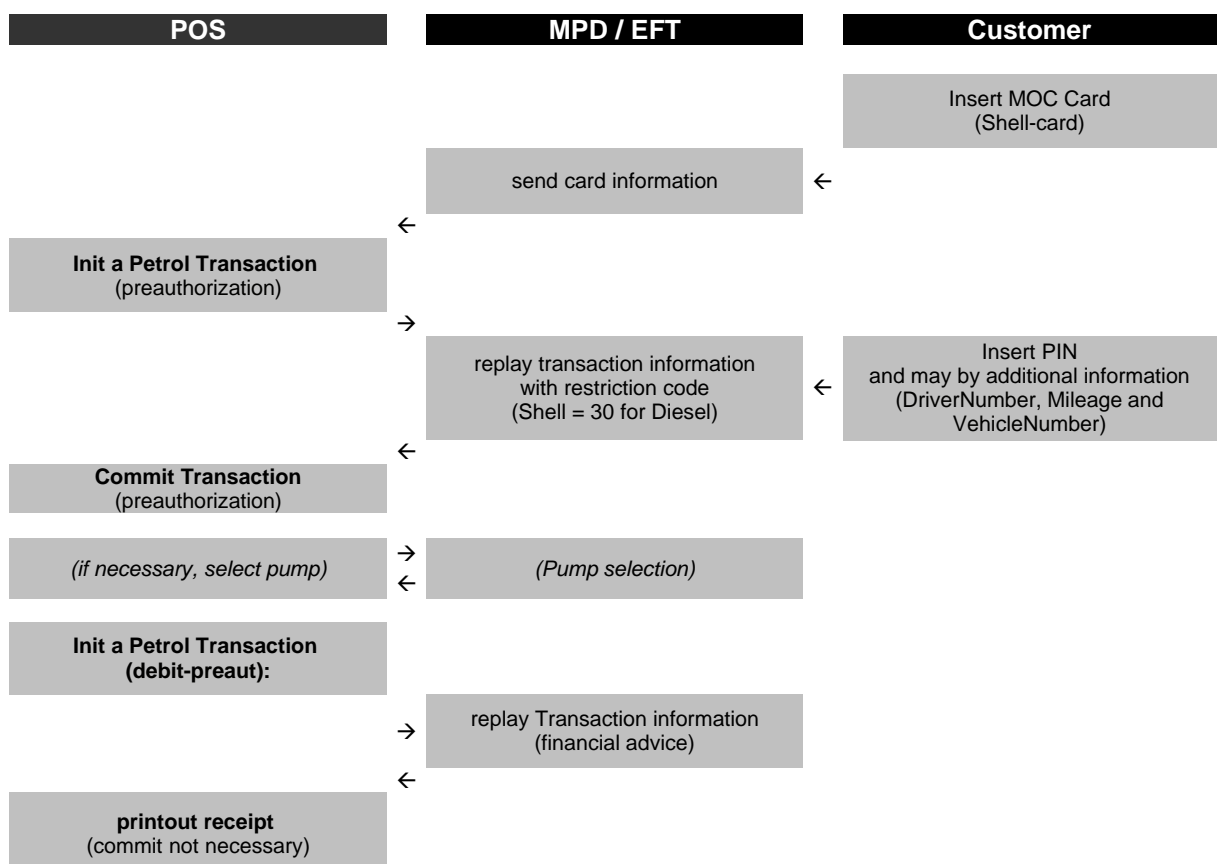
- If the option ProceedNeeded is set, the transaction can be handled by EFT or ECR.
- Processed by EFT = "normal case". Sales Item can be added after recognizing the card. Time limit for continuation (proceed) of the transaction is 5 seconds.
- Processed by ECR = "special case". A transaction response will be provided "EFT\_E\_END\_Silent". At this point the ECR has the control. Only then a PIN check can be requested.
- Further information can be found in the "**EFT Terminal Framework / IFSF EPS Interface**" specification.

**Indoor Transaction Flow**

example flow of indoor petrol transaction, with ProceedNeed: The indoor petrol transaction flow must be executed in asynchrony mode.

**Outdoor Transaction Flow**

example flow of outdoor petrol transaction:



### 3.12.8 Petrol Sale Item

#### Petrol Sale Item

Some card schemes require the submission of sale item information along with financial transaction. The sale item can be set before the transaction (indoor = "[debit](#)" / outdoor = "[preauthorisation](#)"). The application calls the method "WriteProperty" to set the sale item (refer to [WriteProperty Method](#)). Each sale item has a one id (IFSFSalesItemid).

After the transaction and before the commit, the restriction codes can be read (refer to [ReadProperty Method](#)).

```
WriteProperty (table, index, value)
```

#### Table Element

EFT_PE_IFSFProductCodeScheme	string	mandatory	Defines the product code (Shell = "99" / DKV = "98" / etc.)
EFT_PE_IFSFSalesItemid	string	mandatory	Number of the saleitem ("a001" / "a002" / etc.)
EFT_PE_IFSFProductCode	string	mandatory	Product code (Shell = "030" for Diesel" / DKV = "0009" for Diesel)
EFT_PE_IFSFSalesItemAmount	string	mandatory	Amount of the saleitem
EFT_PE_IFSFUnitMeasure	string	mandatory	Unit of measure (liters = "L" / weight = "W" / units = "U")
EFT_PE_IFSFUnitPrice	string	optional	Unit price (eg. "1.46" per liter Diesel)
EFT_PE_IFSFQuantity	string	optional	Quantity (eg. "61.11" of Diesel)
EFT_PE_IFSFTaxCode	string	mandatory	This defines the VAT (value added tax) category.
EFT_PE_IFSFAdditionalProductCode	string	optional	This is the article number from the POS article database.

After the transaction and before the commit, the restriction codes can be read (refer to [ReadProperty Method](#)).

```
ReadProperty (table, index, value)
```

#### Table Element

EFT_PE_IFSFRestrictionCodes	string	The allowed product codes are returned. (Shell-Diesel = "030" // Shell-Unleaded = "022")
EFT_PE_IFSFDriverNumber	string	The element is present if requested during authorization
EFT_PE_IFSFMileage	string	The element is present if requested during authorization
EFT_PE_IFSFFleetID	string	The element is present if requested during authorization

#### Remarks

- There are MOC cards for which certain products are not permitted.
- **Indoor:** If a transaction with one sales item is declined, no RestrictionCodes is available. If a transaction with more than one sales items is declined, RestrictionCodes are available for the approved products. A new transaction must be started with the approved SalesItems.
- **Outdoor:** The pre-authorization transaction contains the approved products (RestrictionCodes). The approved products can be purchased.
- The DriverNumber, Mileage and VehicleNumber elements are present when the card was processed by the EFT and the respective value were requests before online authorization as defined by the card profile used to process the card.
- Product codes may vary. Ensure that the correct format is sent. Example Shell = Product code must be sent in 3 digits (Diesel = "030").

## 3.13 Loyalty Transaction Operation

#### Loyalty Transaction Operation

a Loyalty Transaction gives some additional information about the cardholder. This option is only possible on ep2 terminals, when the Loyalty option is configured on this terminal.

The transaction is done in two phases:

- phase 1: get Loyalty information
- phase 2: continue transaction.

The transaction can also be done in one phase, then the Loyalty information transaction is available at the at the end of the transaction. Both phases are processed by a transaction method call, the flow is controlled by the [LoyaltyMode Property](#). For more information about Loyalty information see [LoyaltyMode Properties](#)

Loyalty transaction flows:

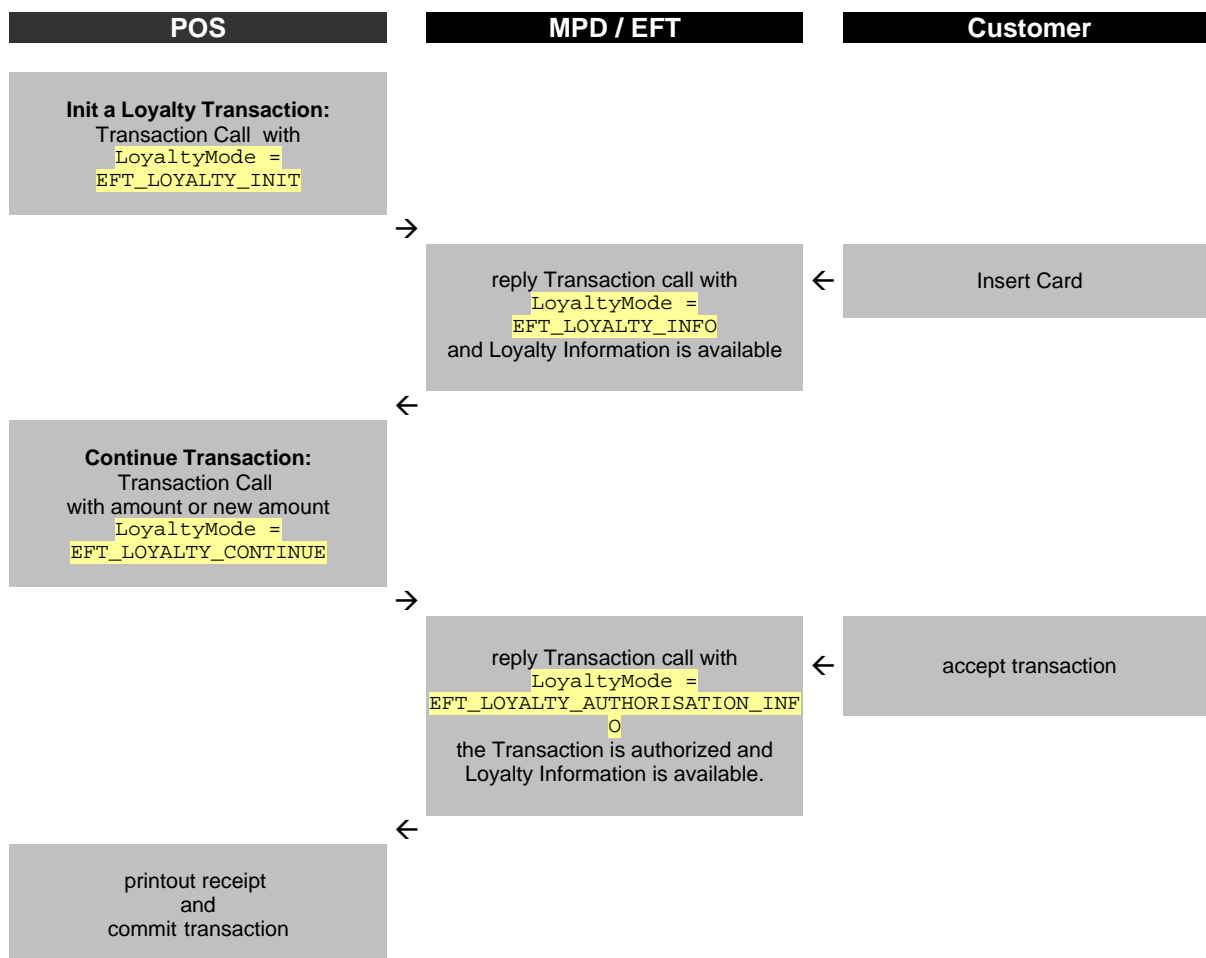
[Loyalty Transaction flow: with Loyalty information](#)  
[Loyalty Transaction flow: without Loyalty information](#)

#### Remarks

- the transaction can be aborted before a card is inserted and also in PHASE\_1\_OK state with **eft.Abort()**
- errors during the transaction are reported in the same way, as in a normal transaction.
- the driver is in the EFT\_S\_PREPARED state during the PHASE\_1\_OK state
- Loyalty information are available in the [LoyaltyData Property](#)

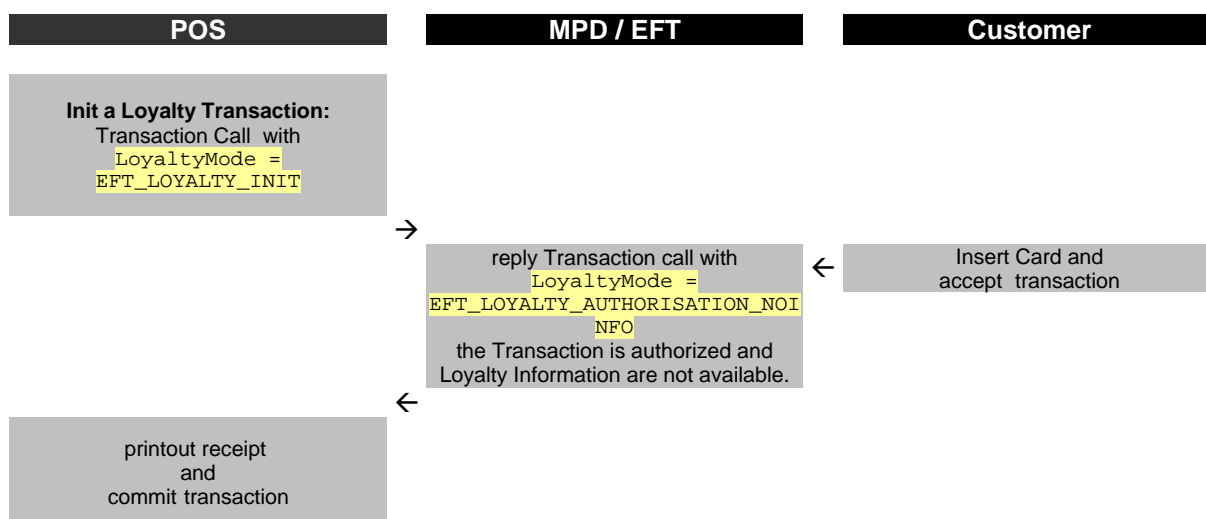
### 3.13.1 Loyalty Transaction flow: with Loyalty Information

flow of a Loyalty Transaction with Loyalty Information:



### 3.13.2 Loyalty Transaction flow: without Loyalty Information

flow of a Loyalty Transaction with without Loyalty Information:



## 4 Runtime Library

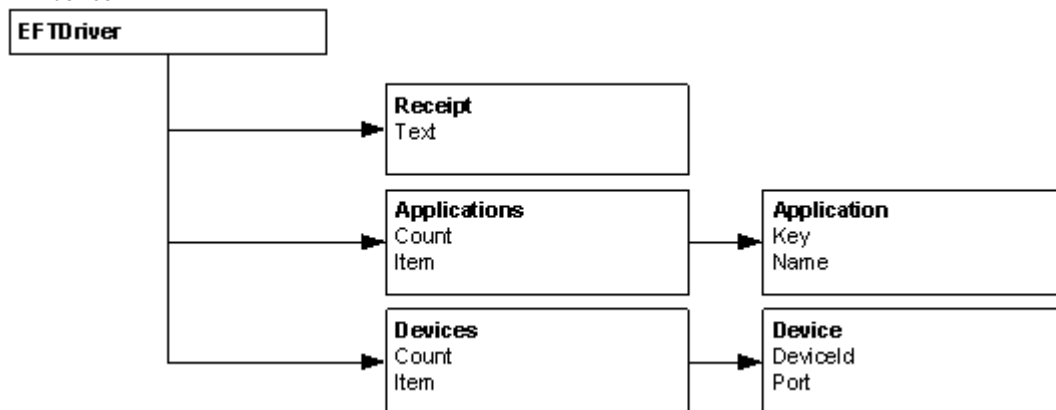
### 4.1 Runtime Library

#### Runtime Library

The runtime library implements the client side of the EFT Control Protocol and is available in C, COM or Java versions. The interface structure is identical on all platforms, there are specific differences however.

#### Object Model

The EFT driver object model is centered on a single component, the Driver object. It represents a client session with the EFT device.



#### See Also

[Driver Object](#)  
[Applications Collection](#)  
[Application Object](#)  
[Receipt Object](#)  
[Devices Collection](#)  
[Device Object](#)

### 4.2 Error Codes and Messages

#### Error Codes and Messages

The following exception codes are used by the driver to indicate various failure conditions.

Error Code	Message	Description
<i>device errors</i>		
101	data-link timeout	The driver failed to connect to the device because of a data-link failure. Check that the EFT device is properly attached to the computer running the driver and that it supports the requested protocol.
102	data-link reject	The device rejected the connection request from the driver. It might currently not be able to handle operations.
103	socket-link reset	The socket link was reset.
201	device not available	No device can handle the requested operation. This may happen when no device is connected to the local computer.
202	device used by other ecr	No device can handle the requested operation. This may happen when the device is already used by another session/ECR.
<i>session errors</i>		
301	session busy	The current session is busy with another operation.
302	session not connected	The session is not connected to a device.
303	invalid request	An invalid request was sent to the driver.
305	invalid arguments	Some parameter of the last request had invalid values.
307	invalid state	The current session is not in the correct state for processing the request. This may happen if you call anything else than Open the first time.
308	unsupported device command	this device command is not supported.
<i>driver errors</i>		
400	generic driver error	Generic driver error.
401	application timeout	The EFT device did not respond to a request within the expected time.

405	operation aborted	The operation as aborted by the ECR using the Abort method
406	format error in request	The request data have a format error
407	unsupported currency	The requested currency is not supported by the driver/protocol
408	driver busy	The driver is busy with another operation
409	journal access failed	The driver failed to access the journal file
410	close shift	Close the shift to perform the requested operation
411	ECR Timeout	The ECR system did not acknowledge a message in the given time
420	general error	General error

*terminal*

501	client not ready	Client has not inserted card
503	card removed	The card has been removed from the reader.
504	shift closed	The shift is currently closed. Perform the Open operation first.
505	remove preceding card	A card from a preceding transaction is waiting to be removed
506	try later	
507	amount zero	
508	not allowed	
509	wrong cashier	
510	wrong ecrId	
511	wrong print mode	
512	wrong terminal	
513	need balance	Perform Balance-Operation to continue.
514	request failed	
515	unsupported function	Unsupported device-control function.
516	fill in voucher	
517	control command failed	
518	ICC command timeout	
519	ICC no card	
520	format error in reply	the reply from the terminal has a format error
521	terminal busy	the terminal is busy, try later

*authorization errors*

600	authorization failed	The authorization failed because of a technical reason.
601	authorization denied	An transaction was aborted because authorization was denied by the acquirer.
602	limit exceeded	Transaction was denied because cardholder limit was exceeded.
603	double operation	Transaction has been executed with identical sequence number.
604	cardholder abort	Cardholder pressed STOP key.
605	cardholder timeout	Cardholder did not insert card within protocol specific timeout.
606	reversal invalid	An invalid reference number was passed.
607	shift closed	The shift is currently closed, perform the <a href="#">Open</a> operation before attempting transactions.
610	unknown transaction	Transaction unknown.
611	cash or other card	Use cash or other card.
612	cash pick-up refused	
613	cash limit exceeded	
614	fill-in voucher	Perform transaction manually.
615	pick-up card	Pick up the card.
617	referral	contact the issuer bank, phone number is in ReferralPhoneNumber
618	invalid track data	

*client component errors*

801	client session busy	The session is busy with another operation.
802	application exception	An exception has occurred while processing the command
803	client session timeout	The client failed to receive an answer from the driver process within 5 minutes.
804	transaction not prepared	This happens when Commit is called without a prior Transaction.
805	socket problem	A problem occurred on the connection to the driver process.
811	receipt unavailable	There is currently no receipt available.
812	invalid attribute	Attempt to request an non-existing attribute
813	application information unavailable	Application information was not yet received from the driver process.
814	client socket closed	The connection to the driver process was closed.
815	failed to connect to driver process	Probably the driver process is not started
816	device information is not available	
817	buffer overflow	
818	function is not implemented	

## 4.3 Timeouts

**Timeouts**

The following table lists the timeouts for the main functions of the EFT system in seconds. They are configured in the driver binary.

Function / Protocol	VEZ	KESS 1.0	KESS ep2
---------------------	-----	----------	----------



<b>Open</b>	30		
<b>Close</b>	30		
<b>Transaction</b>	60		
<b>Balance</b>	60		
<b>DeviceControl</b>	30	n/a	n/a

## 4.4 Driver Object

### 4.4.1 Driver Object

#### Driver Object

##### Methods

Methods are executed synchronously, when the [Async](#) property is set to *false*. Otherwise they run in the background.

<a href="#">Abort</a>	Tries to abort current operation.
<a href="#">Balance</a>	Performs balance operation.
<a href="#">Connect</a>	Connects to the driver process.
<a href="#">Disconnect</a>	Disconnects the driver process..
<a href="#">Open</a>	Perform the shift-open operation on the attached EFT device.
<a href="#">Close</a>	Perform the shift-close operation.
<a href="#">Commit</a>	Commit or rollback after previous, successful Transaction call.
<a href="#">CommitPartial</a>	Partial Commit for vending machines with more than one product per sale
<a href="#">Complete</a>	Called to complete an asynchronous operation.
<a href="#">DeviceControl</a>	Used to perform Initialization, Submission and other, device specific implementations.
<a href="#">QueryStatus</a>	Query EFT device status.
<a href="#">Transaction</a>	Starts transaction operation
<a href="#">Validation</a>	Validates transaction request. This function may be used to test if a particular operation could be executed.

A method call can fail if the current session is not in a valid state, a device or network error occurs or an operation is rejected. It will report such a condition on a language specific mechanism, such as exceptions or return values. The timeouts are different for each function and depend on the device type and the protocol. See [timeouts](#) for more information.

##### Properties

Behaviour configuration and optional parameters are set through properties.

<a href="#">ApplicationKey</a>	read	Returns the key of the application used to run the preceding transaction on the EFT.
<a href="#">Applications</a>	read	Returns a collection of applications currently configured on the device.
<a href="#">Async</a>	read/write	Controls asynchronous operation mode: If true, method calls return immediately and operations must be completed with the Complete method. The default value is false.
<b>AuthCode</b>	read	Transaction Authorization Code
<b>AuthText</b>	read	Transaction Authorization Text
<b>AID</b>	read	Application Identifier (valid for EMV Terminals)
<b>BookingPeriod</b>	read	Transactions booking period
<b>CardRemoveIndicator</b>	read/write	Indicates whether an additional application will be started after the transaction and the card shall not be removed. 0 = Eject Card after transaction (default) 1 = Retain card after transaction <i>Remark: this field shall be set before the transaction will be started</i>
<a href="#">Cashier</a>	read/write	Configures cashier code
<a href="#">CardExpiration</a>	read	Card expiration date, format MMY.
<b>CardNumber</b>	read	Card number from preceding transaction
<b>ContractNumber</b>	read	Merchants Contract Number of preceding transaction
<b>CommittedAmount</b>	read	shows the booked amount after Commit
<b>Currency</b>	read/write	Select transaction currency. The default is CHF: use ISO 4217 alphanumeric codes.
<a href="#">ECRIId</a>	read/write	Optional identifier for the ECR.
<b>ECRSequence</b>	read	Returns ECR sequence number of the last operation.
<a href="#">ExceptionCode</a>	read	The code of the last exception that occurred in the current session.
<a href="#">ExceptionMessage</a>	read	Error message of the last exception.
<b>Id</b>	read/write	Optional ECR specific identifier for the operation. All responses associated with a request will have the same value. The application can use this value for its own purposes. If possible, it will be passed to the EFT device.
<a href="#">Language</a>	read/write	Sets the language the EFT device should use for the GUI. The driver will also use this setting when generating receipts and error messages.
<a href="#">NativeStatus</a>	read	Native status value returned from the EFT device. This value is depending on the protocol used.
<a href="#">NativeMessage</a>	read	Native message value returned from the EFT device.
<a href="#">OperationMode</a>	read/write	Optionally selects EFT operation mode

<a href="#">PrinterWidth</a>	read/write	Set the printer width to be used with automatic receipt generation. This property must be set before the open operation to take effect on subsequent transactions. Possible values are 24 (default) and 40.
<a href="#">Receipt</a>	read	Retrieve the automatically generated receipt after an operation or read particular receipt fields.
<a href="#">ReceiptCopyCount</a>	read	Returns number of receipt copies to print.
<a href="#">ReceiptOptions</a>	read/write	Sets or retrieves the receipt printing option value.
<a href="#">RefNumber</a>	read/write	Reference number of transaction. Must be set by caller with some operations.
<a href="#">Status</a>	read	Returns the current session status.
<a href="#">Time</a>	read	Returns date and time the last operation took place.
<a href="#">Track2</a>	write	Contains track-2 data. This parameter is passed to the EFT device when a transaction is started with <a href="#">EFT_XOPT_USE_TRACK2</a> . <b>Remark:</b> to be out of PCI Scope only technical cards beginning with <b>9</b> can be used
<b>Track3</b>	write	Used to specify track-3 data.

Most properties write access is restricted to idle session states. A runtime error will be raised, when an attempt to modify property values in a busy state.

#### Events

The runtime library informs the application of events that occur during processing operations.

<a href="#">OnStatusChange</a>	Called when session status changes.
<a href="#">OnCommandComplete</a>	Called when operation is ready to be completed immediately.
<a href="#">OnDeviceEvent</a>	Called when a card is inserted into the EFT device.
<a href="#">OnError</a>	Called when an exception situation occurs.

## 4.4.2 Methods

### 4.4.2.1 Open Method

#### Open Method

The application calls this method to initiate a shift-open operation on the attached EFT device.

```
eft.Open ( )
```

#### Properties in

DeviceId	integer	optional	Id of the Terminal. see <a href="#">DeviceId Property</a>
AutoMode	integer	optional	See <a href="#">AutoMode Property</a>
ECRId	string(8)	optional	Identifier of the ECR. see <a href="#">ECRId Property</a>
Cashier	integer	optional	Specifies cashier number for the shift. see <a href="#">Cashier Property</a>
TerminalId	string(12)	optional	[VEZ, KESS] attribute ignored. [EP2] specifies the ep2:TrmID attribute
OperationMode	string(12)	default (interactive)	(default) use interactive device for transactions see <a href="#">OperationMode Property</a>
PrinterWidth	integer	default(24)	Printer width for receipt generation( 24 to 80). see <a href="#">ReceiptOptions Property</a>
Language	string(4)	optional	see <a href="#">Language Property</a>
ReceiptOptions	integer	optional	see <a href="#">ReceiptOptions Property</a>
EcrManufacturer	string(32)	optional	POS Application Manufactor Name. see <a href="#">EcrManufacturer Property</a>
EcrName	string(32)	optional	Name of the POS Application. see <a href="#">EcrName Property</a>
EcrVersion	string(32)	optional	Version of the POS Application. see <a href="#">EcrVersion Property</a>
EftModuleManufacturer	string(32)	optional	EFT Module Application Manufactor Name. see <a href="#">EftModuleManufacturer Property</a>
EftModuleName	string(32)	optional	Name of the EFT Module. see <a href="#">EftModuleName Property</a>
EftModuleVersion	string(32)	optional	Version of the EFT Module. see <a href="#">EftModuleVersion Property</a>

#### Properties out

ECRId	string(8)		Identifier of the ECR as supplied with the associated request
ECRSeq	integer		Transaction Sequence number maintained by the driver:
Actseq	integer	optional	[EP2] derived from the ep2:POSPTrxSeqCnt Attribute Activation Sequence Number: [VEZ, KESS] ignored [EP2] derived from ep2:ActSeqCnt
TerminalId	string(12)		Terminal Identification key. [VEZ, KESS] always supplied by EFT device. [EP2] from request
MpdVersion	string(12)		Version of MPD Server
Time/date			Date and time of operation at EFT. see <a href="#">Time Property</a>
ReceiptCopyCount	integer		see <a href="#">ReceiptCopyCount Property</a>
Receipt	string		Preformatted receipt text of the Shift open receipt. see <a href="#">Receipt Property</a>

**Error Conditions**

If the session is already busy with another operation, this method fails. It fails if no suitable device is available. See [error codes](#) for a list of possible values.

**4.4.2.2 Close Method****Close Method**

Perform shift-close operation.

```
eft.Close ( )
```

**Properties in**

DeviceId	integer	optional	Id of the Terminal. <a href="#">DeviceId Property</a>
AutoMode	integer	optional	See <a href="#">AutoMode Property</a>
ECRId	string(8)	optional	Identifier of the ECR. see <a href="#">ECRId Property</a>

**Properties out**

ECRId	string(8)	Identifier of the ECR as supplied with the associated request.
ECRSeq	integer	Transaction Sequence number maintained by the driver: [EP2] derived from the ep2:POSPTrxSeqCnt attribute see <a href="#">ReceiptCopyCount Property</a>
ReceiptCopyCount	integer	
Receipt	string	Preformatted receipt text of the Shift close receipt. see <a href="#">Receipt Property</a>

**Error Conditions**

If the session is already busy with another operation, this method fails. It fails if no suitable device is available. See [error codes](#) for a list of possible values.

**4.4.2.3 Transaction Method****Transaction Method**

The application calls this method to initiate a new transaction operation on the EFT device. see also [Transaction Operation](#)

```
eft.Transaction ( FunctionCode, Amount , Options )
```

**Parameters***FunctionCode*

Specifies the type of transaction.

<b>debit</b>	A debit transaction shall be started: The cardholder transfers money to the merchant.
<b>debit-wCB</b>	A debit transaction with CashBack shall be started: The cardholder gets goods and CASH money. The AmountOther Property defines the CASH amount to be given to the cardholder
<b>cash-advance</b>	A cash advance transaction
<b>mail-order</b>	A mail-order transaction
<b>credit</b>	A credit transaction: Merchant transfers money to cardholder.
<b>reversal</b>	Cancels a previous transaction. The <a href="#">RefNumber</a> property and the Amount must be set to the referenced transaction.
<b>referral</b>	A referral transaction requires the <a href="#">AuthCode</a> to be set.
<b>reservation</b>	A reservation transaction: Make a preauthorization of an amount on a card.
<b>reservation-extend</b>	increase the reservation amount.
<b>reservation-cancellation</b>	cancels a reservation: a reserved amount on the card will be canceled
<b>reserved</b>	Make debit transaction based on previous reservation. The <a href="#">TrxRefNum</a> property must be set to the reservations transaction reference number and AmountOrig to the original amount on reservation
<b>reserved-referred</b>	Make debit transaction based on previous reservation with the authorization-code to be passed in the RefNumber property.
<b>own-risk</b>	An own-risk debit transaction.
<b>topup</b>	refill transaction for mobile prepay (see <a href="#">Topup Properties</a> )
<b>topup-reversal</b>	reversal of the previous topup transaction
<b>card-activation</b>	Fill a new balance on a Gift Card (see <a href="#">GiftCard Transaction</a> )
<b>card-activation-reversal</b>	Reversal of card activation
<b>card-balance</b>	Return the balance of the card (in property Saldo)

<b>load</b>	load an amount to a Gift Card (ep2: Giftcard / ValueMaster)
<b>load-reversal</b>	Reversal of load
<b>buy</b>	buy points
<b>buy-reversal</b>	Reversal of buy points
<b>collect</b>	collect points
<b>collect-reversal</b>	Reversal of collect points
<b>coupon</b>	coupon transaction (see <a href="#">Coupon Transaction</a> )
<b>coupon-reversal</b>	Reversal of a previous coupon transaction (see <a href="#">Coupon Transaction</a> )
<b>preauthorization</b>	Preauthorization of an amount for the Petrol environment.
<b>debit-preaut</b>	Make debit transaction based on previous preauthorization. The <a href="#">RefNumber</a> property and the Amount must be set to the referenced preauthorization.

**Amount**

Transaction amount in minor currency unit, not containing any separators, dots etc.

**Options**

An options combination of flags that control the use of track-information with the transaction. See [Transaction Options](#) for details. The default is zero and indicates that no track data is used.

**Properties in**

DeviceId	string(8)	optional	Id of the Terminal. see <a href="#">DeviceId Property</a>
Currency	string(3)	optional	Currency: ISO 4217 alphanumeric code, default CHF
Track2	string(40)	optional	Track2 data from external reader or manual input, see <a href="#">Track2 Property</a>
EntryMode	string(1)	default("A")	Set to "M" for manual input of Track2 data.
CVC	string(10)	optional	Card verification code 2 for manual card data input.
Track3	string(108)	optional	Track3 data from external reader
Options	integer	default(0)	See <a href="#">Transaction Options</a> for possible values.
RefNumber	string(11)	optional	Reference number for reversal and reservation-extend transactions see <a href="#">RefNumber Property</a>
TrxRefNum	string(11)	optional	reservations transaction reference number for reserved transaction. see <a href="#">TrxRefNum</a>
AmountOrig	integer	optional	original amount for reserved transactions
AuthCode	string(6)	optional	referral transactions require the <a href="#">AuthCode</a> to be set.
Language	string(4)	optional	See <a href="#">Language Property</a>
TrxTimeout	integer	default(60)	Timeout until the card has to be inserted, in seconds > 60. A value < 60 will indicate to use the terminal default value for the card handling.
CardRemoveIndicator	Integer	default(0)	if 1, the card is not removed after a transaction. It must be removed then with the DeviceControl method. This is useful for ICC operations after a transaction.
DCCMode	integer	default(0)	default: no DCC See <a href="#">DccMode Property</a>
AmountOther	integer	optional	additional amount to be authorized see <a href="#">AmountOther Property</a>
AccountIndex	integer	optional	account index see <a href="#">AccountIndex Property</a>
ECRSeqCounter	string(8)	optional	account index see <a href="#">ECRSeqCounter Property</a>
LoyaltyMode	integer	default(0)	default: no Loyalty See <a href="#">LoyaltyMode Property</a>
LoyaltyInfo	string(20)	optional	Loyalty information. see <a href="#">LoyaltyInfo Property</a>

**Properties out**

Amount	integer		Authorized amount. <b>Remark:</b> if the transaction was tipped by the cardholder (EFT >= ep2 V4.1) then the authorized amount will be higher then the requested amount
Actseq	integer	optional	Activation Sequence Number: [VEZ, KESS] ignored [EP2] derived from ep2:ActSeqCnt
AuthCode	string(6)	optional	Authorization Code [EP2] ep2:AuthC
AuthText	string(64)	optional	Authorization text
RefNumber	string(11)	optional	Reference number of the transaction see <a href="#">RefNumber Property</a>
TrxRefNum	string(11)	optional	reservations transaction reference. see <a href="#">TrxRefNum</a>
ECRSeq	integer		ECR Sequence maintained by driver
Time	date		Date and time of operation at EFT. see <a href="#">Time Property</a>
CardNumber	string(32)	optional	[EP2] ep2:AppPAN
CardExpiration	string(4)	optional	[EP2] ep2:AppExpDate. <b>Remark:</b> since ep2 V5.0 (PCI) this field is optional.
ContractNumber	string(32)	optional	
TerminalId	string(12)	optional	
AID	string(64)	optional	[EP2] ep2:AID
ApplicationName	string(64)	optional	
ApplicationLabel	string(64)	optional	Application Label shown on the Receipts see <a href="#">Application Label Property</a>
ApplicationKey	string(20)	optional	
BrandId	integer	optional	unique Brand identifier, see <a href="#">BrandId Property</a>

BookingPeriod	integer	optional	
EncryptedPan	string()	optional	
ReferralPhoneNumber	string()	optional	
PosEntryMode	string(2)	optional	[EP2] ep2:PosEntryMode. For other devices: 'A' or 'M'. <b>Remark:</b> since ep2 V5.0 (PCI) this field is optional.
SaldoFlag	boolean		true, if the authorisation includes a balance
Saldo	integer	optional	balance on a gift card
ReceiptCopyCount	integer		see <a href="#">ReceiptCopyCount Property</a>
Receipt	string		Preformatted receipt text
TipAmount	integer		see <a href="#">TipAmount Property</a>
CustomerLanguage	string(4)	optional	see <a href="#">CustomerLanguage Property</a>
NbrInstalments	integer	optional	see <a href="#">NbrInstalments Property</a>
AmountOther	integer	optional	see <a href="#">AmountOther Property</a>
FeeAmount	integer	optional	see <a href="#">FeeAmount Property</a>
AcqID	string(11)	optional	see <a href="#">AcqID Property</a>
StaticKeyPanIndex	string(2)	optional	see <a href="#">StaticKeyPanIndex Property</a>
SubBrand	string(20)	optional	see <a href="#">SubBrand Property</a>
LoyaltyMode	integer		see <a href="#">LoyaltyMode Property</a>
LoyaltyData	string(20)	optional	Loyalty information. see <a href="#">LoyaltyData Property</a>

### Error Conditions

This operation will fail the authorization is declined by the acquirer system. It will also fail, if the device is busy with another operation. See [error codes](#) for a list of possible values.

### Remarks

It is necessary to call the [Commit](#) method within a short time after completion of a successful transaction operation to make it persisted. Otherwise the EFT device will abort the operation and financial flow will not take place.

A "Inittransaction" without an amount can be performed by using the Transaction Method with amount 0. This makes the terminal ready to load a card (e.g. opens shutter, peeps, illuminates display). a new Transaction call can then be made with the correct amount (e.g. after a card is loaded).

For DCC transactions (direct currency conversion) see [DccMode Property](#)

## 4.4.2.4 Transaction Options

### Transaction Options

The following codes may be combined to specify options for a payment transaction started with the [Transaction](#) method:

EFT_XOPT_USE_TRACK2	Pass <a href="#">Track2</a> and EntryMode values to EFT device to start a transaction with manually entered data.
EFT_XOPT_USE_TRACK3	Pass Track3 value to EFT device
EFT_XOPT_TIPPABLE	Make transaction tippable. Only debit-transactions can be tipped.
EFT_XOPT_PARTIALCOMMITTABLE	This Transaction will be committed with the <a href="#">CommitPartial Method</a> . This option should only be set on a vending machine, when more than one product has to be given out.
EFT_XOPT_DEV_PANKEYENTRY	the EFT Device shows a dialogue to enter manually the card data (only if supported by the EFT device)
EFT_XOPT_PARTIALAPPROVAL_POSSIBLE	the EFT Device is allowed to authorize with an partial or smaller amount than requested

Defined in enum EFTTransactionOption

## 4.4.2.5 Commit Method

### Commit Method

Perform commit-operation after a successful Transaction call.

```
eft.Commit ( Outcome )
```

### Parameters

*Outcome*

If *true* the transaction will be committed, if *false*, it will be rolled back.

### Properties

after the Commit has returned with no errors::

*Outcome*

If *true* the transaction was committed. (if a rollback was requested, but is not possible, *Outcome* is also *true*)

If *false* the transaction was rolled back.

*CommittedAmount*

the booked amount. on rollback this is 0. on rollback after CommitPartial this is the amount what was really booked.

#### 4.4.2.6 CommitPartial Method

##### CommitPartial Method

Performs a partial commit for vending machines which give out more than one product in a vending process.

```
eft.CommitPartial ( CommitPartialAmount )
```

after give out of each product, a CommitPartial call has to be done. After the last product or a fail, the eft.Commit method has to be called. call eft.Commit(false) if failed.

behaviour on KESS:

After the first CommitPartial call, the full amount is committed on the terminal.

after the last product or a fail, the eft.Commit method has to be called. In case of Commit false, a credit operation is done on the terminal for the amount which was not committed till now and a receipt is generated with the debit and the credit operation.

behaviour on VEZ+:

for each CommitPartial call, the timeout on the terminal is reseted.

In case of Commit false, the commit on the terminal is done with a different amount (reduced to the amount which was committed by partialCommits) and a receipt is generated with the corrected amount.

##### Parameters

###### *CommitPartialAmount*

The value of the product, which was given out

###### *CommitPartialTimeout*

This property can be set before the call to set the timeout until the next PartialCommit or Commit (in seconds, default is 60sec).

For the first CommitPartial call, the timeout is always 60sec from the success of the Transaction call

##### Affected Properties

###### *Amount*

After completion of CommitPartial, Amount shows the amount which was not committed till now.

##### Restrictions

only Transactions with [Option](#) EFT\_XOPT\_PARTIALCOMMITTABLE can be committed with CommitPartial.

#### 4.4.2.7 Balance Method

##### Balance Method

Performs the balance-operation on the attached EFT device.

```
eft.Balance ( )
```

the balance values are stored in the application objects.

After the balance, the following property is set: **BalanceTime**

##### Remarks

The application may perform the balance operation at any time, independently from the shift state. Some devices may however deny balance operation in certain states. Also may the balance operation itself influence the shift state of the device (by e.g. closing it). With **amodeBalanceFromDriver** the BalanceCounters are calculated by the driver, instead read from the EFT-Device.

#### 4.4.2.8 Validation Method

##### Validation Method

The application calls this method to test if a particular transaction operation might be started on the EFT device.

```
eft.Validation ( FunctionCode, Amount [, Options ] )
```

##### Remarks

This function takes the same parameters as the Transaction method. It calls the driver to test, if the requested operation,

such as a reversal, can be performed. If yes, the function will succeed, otherwise an error occurs. There is, however, no guarantee, that the transaction will really succeed, since the EFT-device may still decide to deny it.

#### 4.4.2.9 Abort Method

##### Abort Method

###### Summary

During a transaction:

- Enters the abort sequence for an transaction operation. The effect of this method is reported through the result code of the aborted Transaction operation.

During a card is loaded before a transaction:

- Enters the abort sequence for loaded card before transaction. The effect of this method is reported through the OnDeviceEvent with new DeviceEventCode.

```
eft.Abort ( )
```

##### Remarks

a transaction can be aborted until a card is entered into the reader. if an abort is sent afterwards, the transaction will be done, but automatically refused by the driver (except CASH-transactions, which cannot be refused).

#### 4.4.2.10 QueryStatus Method

##### QueryStatus Method

Performs a status request on the EFT device and reports changes of [DeviceEventCode](#), [DeviceStatusCode](#) and [ActivationState](#) with the [OnDeviceEvent-Event](#) .

```
eft.QueryStatus ( )
```

See also [AutoMode Property](#)

#### 4.4.2.11 DeviceControl Method

##### DeviceControl Method

Performs a driver-control operation. It may be used to perform Initialization and Submission operations.

```
eft.DeviceControl ( controlClass, controlCommand )
```

##### Class and Command Codes

Retract Card	0x01 0x01	Retract card into internal storage (vending machines)
Card Reposition	0x01 0x02	Positions the card again
Reader active	0x01 0x03	Activates the card reader (vending machines)
Reader inactive	0x01 0x04	Deactivates the card reader (vending machines)
Eject Card	0x01 0x05	Releases/Ejects the card
Reserved	0x01 0x2X	Reserved or reserved for further use
Reserved	0x01 0x30	Reserved
Setup Configure	0x02 0x01	Runs Configuration operation on the EFT device.
Setup Initialize	0x02 0x02	Runs Initialization operation on the EFT device.
Setup Config and Init	0x02 0x03	Runs Configuration and Initialization operation on the EFT device.
Setup Download	0x02 0x04	Run Download operation (will be started after response)
Diagnosis	0x02 0x05	Display Information from EFT Device (IFSF)
Reboot Device	0x02 0x09	Initiates a device reboot
Reserved	0x02 0x9X	Reserved or reserved for further use
Submission	0x03 0x01	Run PMS submission
ActivateServiceMenu	0x04 0x01	Activate the Service Menu on Terminal (only in closed shift state)
ActivateUI	0x04 0x02	Activate User Interface on Terminal
DCC rates table	0x05 0x01	Print DCC exchange rates table
Topup Voucher Services	0x06 0x01	Available Voucher Services
Display Text	0x08 0x01	Display Text on EFT Device (IFSF)
Display Resource	0x08 0x02	Display a predefined Text on EFT Device (IFSF)



Get Key Input	0x08 0x03	Get Key Input from the EFT Device (IFSF)
Pin Check	0x08 0x04	
Login	0x09 0x01	.
Logout	0x09 0x02	.
Hold	0x09 0x03	hold is used to reset the timeout between authorization and commit(amount)
Driver Query Devicelist	0x20 0x01	Get an updated version of the device list.
Driver Shutdown	0x20 0x02	Shutdown the driver process.
Get Last Init Receipt	0x20 0x03	get the stored receipt from the last initialization
Get Device	0x20 0x04	get the device with the DeviceId. See also <a href="#">DeviceId Property</a>
Release Device	0x20 0x05	release the device
Forced Get Device	0x20 0x07	forced get device with releasing from an already associated Cash Register
Forced Get Device with later Give Back	0x20 0x08	forced get device with releasing from an already associated Cash Register with later return to the previous system
<b>Properties in</b>		
no		
<b>Properties out</b>		
DeviceControlResult	integer	Specifies the command class code
DeviceControlData cases)	string(var)	Additional arguments (currently used for non public cases)
Receipt	string	Preformatted receipt text (optional)
<b>DeviceControlResult Codes</b>		
0x00	operation successful	
0xFE	device control class/command not supported	
0xFF	device control command failed	
0x01	(Get Device Result) the device is connected to another ECR	
0x02	(Get Device Result) the device is not available	

#### 4.4.2.12 DeviceAccess Method

##### DeviceAccess Method

Performs a driver-control operation on Devices on the Terminal.

```
eft.DeviceAccess ( deviceId )
```

##### Properties in

DeviceAccessData	string(var)	Additional arguments (used for non public cases)
------------------	-------------	--

##### Properties out

DeviceAccessResult	integer	Specifies the result
DeviceAccessData	string(var)	Additional arguments (currently used for non public cases)

##### DeviceAccessResult Codes

0x00	operation successful
0xFE	device control class/command not supported
0xFF	device control command failed

#### 4.4.2.13 IccControl

##### IccControl

allows the communication with a integrated chip on card on ep2 terminals with ICC-reader

Method:

```
eft.IccControl ( iccCommandCode )
```

```
typedef enum EFTIccCommandCode {
    EFT_ICC_POWER_UP = 0,
    EFT_ICC_POWER_DOWN = 1,
    EFT_ICC_RESET = 2,
```



```

    } EFTIccCommandCode;
                                EFT_ICC_APDU = 3

```

**Input Properties:**

- IccDeviceId: CCR1 (for the card), CSM1, CSM2, CSM3, CSM4 (in the device)
  - IccAduRequest: APDU command: hex string var length max 530
- Format: CLA INS P1 P2 [Lc Data] [Le]

**Output Properties:**

- IccCardAtr: ATR of the SmartCard: hex string var length max 64
  - IccAduResponse: APDU answer: hex string var length max 530
- Format: [Data] SW1 SW2

**ICC Commands****ICC Power Up****Method**

```
IccControl(EFT_ICC_POWER_UP)
```

**Input Properties:**

```
IccDeviceId
```

**Output Properties:**

```
IccCardAtr
```

**ICC Power Down****Method**

```
IccControl(EFT_ICC_POWER_DOWN)
```

**Input Properties:**

```
IccDeviceId
```

**Output Properties:**

```
no
```

**ICC Reset****Method**

```
IccControl(EFT_ICC_RESET)
```

**Input Properties:**

```
IccDeviceId
```

**Output Properties:**

```
IccCardAtr
```

**ICC APDU****Method**

```
IccControl(EFT_ICC_APDU)
```

**Input Properties:**

```
IccDeviceId
IccAduRequest
```

**Output Properties:**

```
IccAduResponse
```

**Errors**

```
ExceptionCode 305
```

```
EFT_E_INVALID_ARGUMENTS
"invalid arguments"
```

ExceptionCode 308	EFT_E_UNSUPPORTED_DEVICE_COMMAND "unsupported device command" eg: Terminal has no ICC
ExceptionCode 517	EFT_E_DEVICE_COMMAND_FAILED "device command failed" more details in: - NativeStatus : System Error Code - NativeMessage: System Error Description
ExceptionCode 518	EFT_E_ICC_TIMEOUT "icc command timeout"
ExceptionCode 519	EFT_E_ICC_CARD_IS_REMOVED "icc no card"

**Simulation**  
**EFT Simulator**  
 Select Option "has CCRs"  
**ECR Simulator**  
 Press "ICC Commands..."

#### 4.4.2.14 Complete Method

##### Complete Method

The application must call this function to complete a previously started, asynchronous method call.

```
eft.Complete ( Timeout )
```

##### Parameters

*Timeout*

Specifies how long to wait for the completion of the operation in milliseconds

##### Return Value

The function returns successful, if the function call completes within the given time. If the operation does not complete, it fails and the caller may retry the call. If an exception occurs while waiting for completion, the function fails. If Commit end with Timeout, the call fails with exception EFT\_E\_TIMEOUT.

##### Special functionality

```
eft.Complete ( 0 )    use Timeout 0 to inform the driver about completion of the command in async mode
eft.Complete ( 1 )    use Timeout 1 to read status messages from the driver before reading the status fields
                        (only needed for EFTAPI)
eft.Complete ( -1 )   use Timeout -1 to wait for an event, either a DeviceEvent (CompletedDeviceEvent is set),
                        or a CommandCompleted (CompletedCommand is set). After eft.Disconnect():
                        Complete(-1) returns with EFT_E_CLIENT_SOCKET_CLOSED.
```

#### 4.4.2.15 Connect Method

##### Connect Method

The method may be used to connect the runtime to the driver process without performing the open operation immediately. the client connect to the server at the computer with the IP [DriverAddress](#)

#### 4.4.2.16 Disconnect Method

##### Disconnect Method

This method disconnects the runtime from the driver process. A new connection can be opened using the Connect or Open methods.

#### 4.4.2.17 ReadProperty Method

##### ReadProperty Method

Elements, that are stored in the Receipt object can be accessed with the ReadProperty Method.

```
eft.ReadProperty ( table, index )
```

with EFTAPI:

```
char sEftvalue[128];
r = EFT_ReadProperty(h, table, index, sEftvalue, sizeof(sEftvalue));
```

with eftoa COM:

```
Dim sEFTValue as String
sEFTValue = eft.Property(table, index)
```

**with DotNET:**

```
String sEFTValue;
sEFTValue = eft.ReadProperty(table, index);
```

**table:**

```
EFT_PT_TRMInfo = 10,
EFT_PT_BrandInfo = 11,
EFT_PT_ep2sbbRPT = 20,
EFT_PT_postPCD = 21
```

**4.4.2.18 WriteProperty Method****WriteProperty Method**

Elements can be stored in the Extra XML with the WriteProperty Method.

```
eft.WriteProperty ( table, index, value )
```

**with EFTAPI:**

```
char sEftvalue[128];
...
r = EFT_WriteProperty(h, table, index, sEftvalue);
```

**with eftoa COM:**

```
Dim sEFTValue as String
...
eft.Property(table, index) = sEFTValue
```

**with DotNET:**

```
String sEFTValue;
...
eft.WriteProperty(table, index, sEFTValue);
```

**table:**

```
EFT_PT_TRMInfo = 10,
EFT_PT_BrandInfo = 11,
EFT_PT_ep2sbbRPT = 20,
EFT_PT_postPCD = 21
```

**4.4.2.19 PrintOnEFT Method****PrintOnEFT Method**

Performs the print-output on the attached EFT device.

```
eft.PrintOnEFT ( PrintText, Options )
```

*PrintText*

Text to be printed. See [Print Text](#)

*Options*

Options that control the use of the PrintOnEFT operation. See [Print Options](#).

**Properties in**

DeviceId	integer	Id of the Terminal. See <a href="#">DeviceId Property</a>
PrintCharSet	integer	the character set to be used for Printing See <a href="#">PrintCharSet Property</a>
[****]		

**Properties out**

PrinterState	integer	Printer status after a printing request. See <a href="#">PrinterState Property</a>
--------------	---------	--

**4.4.2.19.1 PrintText****PrintText** string(512)

Text to be printed by the EFT.

### Text Formatting

The following table shows all by the EFT supported formatting and printer width.

#### Control Codes:

Name	Description	Code
NORMAL	Normal width, normal high	(/B)
BOLD	Double width, normal high	(B)
SMALL_HIGH	Normal width, half high	(SH)
LEFT	Left adjust	(LA)
CENTER	Center adjust	(CT)
RIGHT	Right adjust	(RA)
BREAK	Carriage Return Line feed	(BR)
HORIZONTAL_ROW	Parting linie	(HR)
FRAME_TOP	Frame headline	(FRMT)
FRAME_LEFT	Frame left border	(FRML)
FRAME_RIGHT	Frame right border	(FRMR)
FRAME_BOTTOM	Frame footline	(FRMB)
PARENTHESIS	Left parenthesis	((
	Right parenthesis	))

#### Printer Width:

Name	Description	Characters
NORMAL	Normal width, normal high	48
BOLD	Double width, normal high	24
SMALL_HIGH	Normal width, half high	48

#### 4.4.2.19.2 Print Options

##### Print Options

The following codes specifies the options for a Printing operation started with the [PrintOnEFT Method](#):

EFT_OPT_PRINT_STATE	Option to obtain the printer state. (set PrintText to "")
EFT_OPT_PRINT_FIRST	Option to indicate the start of a printing job. The print buffer on the EFT will be cleared.
EFT_OPT_PRINT_SUBSEQUENT	Option to indicate next block to print.
EFT_OPT_PRINT_FINAL	Option to indicate last block to print an to finalize the job.
EFT_OPT_PRINT_ABORT	Option to indicate aborting a running print process. (set PrintText to "")

Defined in enum EFTPrintOption

#### 4.4.2.20 CommitAmount Method

##### CommitAmount Method

Perform commit-operation after a successful Transaction call.

```
eft.CommitAmount ( commitPartialAmount )
```

##### Parameters

*commitPartialAmount*

If > 0 the transaction will be committed, if 0 it will be rolled back.

##### Properties

after the CommitAmount has returned with no errors::

##### Outcome

If *true* the transaction was committed. (if a rollback was requested, but is not possible, *Outcome* is also *true*)

If *false* the transaction was rolled back.

##### CommittedAmount

the booked amount. on rollback this is 0. on rollback after CommitPartial this is the amount what was really booked.

### 4.4.3 Properties

#### 4.4.3.1 ActivationState Property

##### ActivationState Property

shows the status of the actual shift in the driver. it changes on Open or Close.

EFT\_SHIFT\_CLOSED The shift is closed

EFT\_SHIFT\_OPEN The shift is open

Defined in enum EFTActivationState

Returns the last value receipt from the driver. See also [OnDeviceEvent](#).

#### 4.4.3.2 ApplicationKey Property

##### ApplicationKey Property

Identifies an EFT application.

Values depend on the underlying protocol:

[VEZ] The string VEZxx where xx is the IssuerNumber is used.

[KES] The string KESxx where xx is the IssuerNumber is used.

[EP2] The ep2:AID is used as identifier.

#### 4.4.3.3 Applications Property

##### Applications Property

###### Summary

Returns and [Applications Collection](#) containing information about EFT applications (card products) available to the current session.

```
eft.Applications
```

#### 4.4.3.4 Async Property

##### Async Property

The application controls through this property the behaviour of method calls. If *true*, all calls will immediately return to the caller and are processed in the background. The application can then check the outcome of the operation by calling the [Complete](#) method. A [OnCommandComplete Event](#) is raised, when the command has completed in async mode.

The default value of this property is *false*. It is not necessary to call complete in the synchronous mode.

##### Command flow in synchron mode:

- Call the Command (eg Open, Transaction)
  - the Command is complete on successfull return
  - the Command has failed on error return

##### Command flow in asynchron mode:

- Call the Command (eg Open)
  - the Command is started on successfull return
  - if the Command could not be started, an error return occurs
- wait for completion with:
  - calling Complete with a Timeout
  - calling Complete(-1), this returns on event (DeviceEvent or Command completed)
  - waiting for the OnCommandComplete Event

#### 4.4.3.5 BrandId Property

##### BrandId Property

Identifies the EFT Brand.

*Remark: please ask for the latest brand id list*

#### 4.4.3.6 CardExpiration Property

##### CardExpiration Property

Returns the expiration date of the card use in the recent transaction as string using format:

MMYY

*Remark: since ep2 V5.0 (PCI) this field is optional.*

#### 4.4.3.7 Cashier Property

##### Cashier Property

This property may be set to configure the current cashier number. Possible values range from 1 to 99999999

#### 4.4.3.8 CompletedDeviceEvent Property

##### CompletedDeviceEvent Property

this property is set to 1 when a status change has occurred on [ActivationState](#) , [DeviceStatusCode](#) or [DeviceEventCode](#) the property is reset to 0 on the next Complete(timeout) call.

#### 4.4.3.9 CompletedCommand Property

##### CompletedCommand Property

this property is set when a command has completed:

```
/* EFTComletedCommandCode */
EFT_CC_OPEN           = 1,
EFT_CC_CLOSE          = 2,
EFT_CC_TRANSACTION    = 3,
EFT_CC_COMMIT         = 4,
EFT_CC_PARTIAL_COMMIT = 5,
EFT_CC_QUERY_STATUS   = 6,
EFT_CC_BALANCE        = 7,
EFT_CC_DEVICE_CONTROL = 8,
EFT_CC_ICC_CONTROL    = 9,
EFT_CC_VALIDATION     = 10,
EFT_CC_DEVICE_ACCESS  = 11,
EFT_CC_PRINT          = 12,
```

the property is reset to 0 on the next Complete(timeout) call.

#### 4.4.3.10 DccMode Properties

##### DccMode Properties

These properties are used to handle the DCC transactions (direct currency conversion) see also [DCCTransactionOperation](#)

##### DccMode Property:

```
0  EFT_DCC_NO_DCC
1  EFT_DCC_INIT_PHASE1
2  EFT_DCC_PHASE1_OK
3  EFT_DCC_CONTINUE_DCC
4  EFT_DCC_CONTINUE_LOCAL
5  EFT_DCC_AUTHORISATION_DCC
6  EFT_DCC_AUTHORISATION_LOCAL
```

The DccMode Property has to be set before the Transaction to:

EFT_DCC_NO_DCC	normal Transaction without DCC (default)
EFT_DCC_INIT_PHASE1	start a DCC phase 1
EFT_DCC_CONTINUE_DCC	start a DCC authorisation phase after a DCC phase 1 ok answer
EFT_DCC_CONTINUE_LOCAL	start a local currency authorisation phase after a DCC phase 1 ok answer

to start a DCC transaction in one phase set the DccMode Property to EFT\_DCC\_CONTINUE\_DCC

The DccMode Property is set to the following value after a successfull Transaction request:

EFT_DCC_NO_DCC	normal Transaction without DCC (default)
EFT_DCC_PHASE1_OK	DCC phase 1 ready, foreign currency information available
EFT_DCC_AUTHORISATION_DCC	the DCC transaction was successfull in foreign currency
EFT_DCC_AUTHORISATION_LOCAL	the transaction was successfull in local currency

##### DCCOriginalDate Property:

this property has to be set for credit operation, when an old debit operation is reversed.

**Format:** YYYYMMDD

#### DCC information properties:

- **DccCurrency:** foreign currency on card of customer
  - **DccAmount:** amount in foreign currency in minor currency unit, not containing any separators, dots etc.
  - **DccAmountExp:** exponent for major currency unit (e.g. 2 if minor units are cents)
  - **DccExrate:** exchangerate: local currency per foreign currency. (this comes as string, because it can be up to 12 digits long)
  - **DccExrateExp:** exponent of DccExrate
- e.g. EUR 1.00 = CHF 1.504174: DccExrate = 1504174, DccExrateExp = 6

#### Remarks

- a successful transaction has to be committed with **eft.Commit(true)**
- the transaction can be aborted before a card is inserted and also in PHASE\_1\_OK state with **eft.Abort()**
- the driver is in the EFT\_S\_PREPARED state during the PHASE\_1\_OK state
- some EFT Terminals aren't able to start with EFT\_DCC\_INIT\_PHASE1 (eg. IFSF eps) in this case the transaction has to be done in one phase DccMode Property = EFT\_DCC\_CONTINUE\_DCC)

### 4.4.3.11 DeviceEventCode Property

#### DeviceEventCode Property

Returns the last value receipt from the driver. See also [OnDeviceEvent](#).

Specifies the type of event:

EFT_READER_EMPTY	The card has been removed from the reader
EFT_READER_LOADED	A payment card has been read or inserted and a new transaction may be started within a device specific timeout. The optional applicationKey argument indicates which application will handle the transaction with the inserted card.
EFT_READER_EJECTED	The card has been ejected from the reader but has not yet been removed from the slot. This event is not generated for manual readers.
EFT_READER_CARD_RETENSION	a card was retained.

Defined in enum EFTDeviceReaderStatus

### 4.4.3.12 DeviceStatusCode Property

#### DeviceStatusCode Property

shows the status of the connected device:

EFT_DEVICE_CLOSED	The device is closed
EFT_DEVICE_OPEN	The device is open
EFT_DEVICE_BUSY	The device is busy and can not accept a command
EFT_DEVICE_NO_ANSWER	There is no answer from the device since 1 min
EFT_DEVICE_NOT_INIT	The device is not detected by the driver till now.
EFT_DEVICE_NO_ANSWER_SEVERE	The TRM gives no answer since 30 min
EFT_DEVICE_PROBLEM_ON_START	The TRM has a Problem during startup (davinci vending only) Additional Info in the property <b>ExceptionMessage</b> In the format "Sx; error text", where x is the TRM error code
EFT_DEVICE_CARD_LEFT	a Card is left in the reader (davinci vending only)

Defined in enum EFTDeviceStatus

Returns the last value receipt from the driver. See also [OnDeviceEvent](#).

### 4.4.3.13 DriverAddress Property

#### DriverAddress Property

This property may be used to configure a remote connection to a driver running on another computer. It takes an IP-Address in dotted notation as argument. The property must be set before a method is called, i.e. when the driver object is in disconnected state.  
also the port number can be set.

#### Example

```
eft.DriverAddress = "192.0.0.13"
```

also the port number can be set by adding a double point and the port number.

**Example**

```
eft.DriverAddress = "192.0.0.13:8165"
```

**4.4.3.14 ECR Information Properties****ECR Information Properties****ECRId Property**

This property is used to set the ECR Identifier passed to the EFT device during operations. The interpretation of the value is specific to the EFT device used. While the property type is string, the range and character set of possible values may be limited by the protocol used.

Allowable values include the range 1.. 999999 of numeric values.

**EcrManufacturer Property**

This property is used to set the ECR Application Manufacturer Information for Customer support purposes

**EcrName Property**

This property is used to set the ECR Application Name for Customer support purposes

**EcrVersion Property**

This property is used to set the ECR Application Version for Customer support purposes

**EftModuleManufacturer Property**

This property is used to set the EFT Module Manufacturer Information for Customer support purposes

**EftModuleName Property**

This property is used to set the EFT Module Name for Customer support purposes

**EftModuleVersion Property**

This property is used to set the EFT Module Version Information for Customer support purposes

**4.4.3.15 ExceptionCode, ExceptionMessage Properties****ExceptionCode, ExceptionMessage Properties****Properties**

ExceptionCode	integer		See <a href="#">Error Codes</a>
ExceptionMessage	string	optional	Textual description of the problem
NativeStatus	string	optional	Native error code received from the EFT device
NativeMessage	string	optional	Native error message received from the EFT device

The two properties contain information about an exception that previously occurred on the current session.

the **ExceptionMessage Property** is set according the selected language.

the ErrorText in a specific language can be received with

```
text = eft.ErrorText(language, exceptionCode)
```

**4.4.3.16 Language Property****Language Property**

Used to configure default languages for the EFT GUI using one or more of the ISO 639 language codes. An invalid value will not cause the operation failure, because it will be handled as English.

Actual supported Languages:



Code	Language	Receipt Text	Error Messages
en	English	x	x
de	German	x	x
fr	French	x	x
it	Italian	x	x
pl	Polish	x	x
hu	Hungarian	x	x
sl	Slovenian	x	x
hr	Croatian	x	x
cs	Czech	x	x
sk	Slovak	x	x
bg	Bulgarian	1)	1)
sp	Spanish	x	1)

1) in English only

#### 4.4.3.17 MPD Version Properties

##### MPD Version Properties

MPDClientVersion: Version of the client part of MPD.

MPDServerVersion: Version of the server part of MPD (this property is set after an open or close command).

#### 4.4.3.18 NativeStatus, NativeMessage Properties

##### NativeStatus, NativeMessage Properties

Allow access to native, protocol and device specific error information. The values may be presented to the operator together with [general error information](#).

#### 4.4.3.19 OperationMode Property

##### OperationMode Property

Specifies processing options for transactions in respect to the acquirer. The setting influences device selection and transaction flags.

<b>interactive</b>	Interactive presentment of card using the EFT device (default).
<b>moto</b>	Make MOTO transactions.
<b>ecom</b>	Make e-commerce transactions.

#### 4.4.3.20 Receipt Properties

##### Receipt Properties

##### new receipt handling:

to switch to the new receipt handling set

**ReceiptOptions = EFT\_RCP\_WITH\_MERCHANT\_RECEIPT**

##### ReceiptOptions Property

Specifies receipt formatting options. This setting will take effect, when the open operation is executed.

EFT_RCP_SUPPRESS_HEADER	Suppress printing of header and footer sections.
EFT_RCP_SUPPRESS_SIGNATURE	Suppress printing of the Signature Text and line if the application prints it itself in case of ReceiptCopyCount = 2
EFT_RCP_SUPPRESS_ECR_INFORMATION	Suppress printing of ECR Information (Attendant-Id, ECR-Id, ECR-Seq)
EFT_RCP_SUPPRESS_EFT_INFORMATION	Suppress printing of EFT Information (EFT Ref-No)
EFT_RCP_COMPACT_FORMAT	use the compact format for transaction receipts
EFT_RCP_SUPER_COMPACT_FORMAT	use the super compact format for transaction receipts
EFT_RCP_ULTRA_COMPACT_FORMAT	use the ultra compact format for transaction receipts
EFT_RCP_WITH_CONTROL_CHARS	receipt has format information (only for topup receipts)
EFT_RCP_WITH_MERCHANT_RECEIPT	new receipt handling: customer receipt and merchant receipt are treated differently
EFT_RCP_SUPPRESS_EFT_DCCDISCLAIMERTEXT	Suppress printing of DCC disclaimer text
EFT_RCP_SUPPRESS_EFT_FEEDISCLAIMERTEXT	Suppress printing of FEE disclaimer text
EFT_RCP_WITH_RECEIPTTYPE_LINE	receipt contains receipt type information like "Cardholder Receipt" and "Merchant Receipt"

**PrinterWidth Property**

Sets the width of printer paper in count of characters. Acceptable values lie within 24 and 80, the default is 24.

**ReceiptText Property**

Returns the receipt from preceding operation as a string to be printed out for the customer.

**ReceiptMerchantText Property**

Returns the receipt from preceding operation for the **merchant**, it can be different from the receipt for the customer.

**ReceiptCopyCount Property**

Returns the number of receipt copies that shall be printed by the ECR applications for the customer. This value may range from 0 to 1.

**ReceiptMerchantCount Property**

Returns the number of merchant receipt copies that shall be printed by the ECR applications for the merchant. This value may range from 0 to 1.

**ReceiptSignatureFlag Property**

Indicates if signature is needed on receipt.

EFT_SGF_NO_SIGNATURE	(= 0) no signature is needed.
EFT_SGF_CUSTOMER_SIG	the customer has to sign on the merchant receipt
EFT_SGF_MERCHANT_SIG	the merchant has to sign on the customer receipt

**old receipt handling:**

this is kept for compatibility reason:

**ReceiptOptions Property**

Specifies receipt formatting options. This setting will take effect, when the open operation is executed.

EFT_RCP_SUPPRESS_HEADER	Suppress printing of header and footer sections.
EFT_RCP_SUPPRESS_SIGNATURE	Suppress printing of the Signature Text and line if the application prints it itself in case of ReceiptCopyCount = 2
EFT_RCP_SMALL_FORMAT	use the small (compact) format for transaction receipts
EFT_RCP_WITH_CONTROL_CHARS	receipt has format information (only for topup receipts)

**PrinterWidth Property**

Sets the width of printer paper in count of characters. Acceptable values lie within 24 and 80, the default is 24.

**ReceiptText Property**

Returns the receipt from preceding operation as a string to be printed out for the customer.

**ReceiptCopyCount Property**

Returns the number of receipt copies that shall be printed by the ECR applications for the customer. This value may range from 0 to 2.

<i>Number of Copies</i>	<i>Transaction Type</i>
0	CASH transactions
1	PIN-based transactions
2	Credit card transactions with signature

**4.4.3.21 RefNumber Property****RefNumber Property**

A EFT device specific transaction identifier that has to be used to cancel transactions and to book tip. This value is usually a six or eight digit decimal number. (ep2: this is the Transaction Sequence Counter). for Stufe 3+ terminals it can also be a hexadecimal representation. it is recommended to handle this property as string(11).

**TrxRefNum Property**

The transaction specific identifier to be used to book or cancel a reserved amount. This is a string (11). (ep2: this is the Transaction Reference Number).

**AuthCode Property**

This property shows the Authorisation Code of an Transaction. For telephonical authorized transactions (referral), this has to be set to the code got by phone. This is a string(6).

**4.4.3.22 Status Property****Status Property**

The Status property is an number that reflects the current EFT session state to the application:

EFT_S_IDLE	Initial state: Session is not connected to the driver. Call the <a href="#">Open</a> method to start a session.
EFT_S_CONNECTION_PENDING	Connection to driver is pending.
EFT_S_CONNECTED	Connection to driver is established, no operation pending.
EFT_S_OPERATION_PENDING	An operation is currently pending on the session.
EFT_S_EXCEPTION	An exception occurred.
EFT_S_PREPARED	The session is prepared to perform the <a href="#">Commit</a> operation.

**4.4.3.23 Time Property****Time Property**

Returns the date and time of the last operation. The value is returned in string format:

yyyymmdd hh:mm:ss

**4.4.3.24 Topup Properties****Topup Properties**

Topup Transactions (PrePay refill, mobile voucher) are done with the Transaction function codes "topup" and "topup-reversal".

**Properties in**

IssuerName	string	Name of the desired issuer for mobile voucher
------------	--------	---

**Properties out**

TopupRefillCode	string	(encrypted)
TopupSerialNumber	string	
ApplicationName	string	name of the prepay brand
CardExpiration	string	Expiration of the refill code
ReceiptText	string	the customer receipt with Refill code and instructions
ReceiptMerchantText	string	receipt for the merchant with transaction informations

**ReceiptText** is preformatted (default) or has format informations (use PeceiptOptions = EFT\_RCP\_WITH\_CONTROL\_CHARS)

control chars:

Name	Description	Code
NORMAL	Normal width, normal high	(/B)
BOLD	Double width, normal high	(B)
SMALL_HIGH	Normal width, half high	(SH)
BIG_HIGH	Double width, double high	(BH)
LEFT	Left adjust	(LA)
CENTER	Center adjust	(CT)
RIGHT	Right adjust	(RA)
BREAK	Carriage Return Line feed	(BR)
HORIZONTAL_ROW	Parting line	(HR)
FRAME_TOP	Frame headline	(FRMT)
FRAME_LEFT	Frame left border	(FRML)
FRAME_RIGHT	Frame right border	(FRMR)
FRAME_BOTTOM	Frame footline	(FRMB)
PARENTHESIS	Left parenthesis	((
	Right parenthesis	))

#### 4.4.3.25 Track2 Property

##### Track2 Property

Used to present Track2 or Track2 equivalent data to the EFT device. In conjunction with EntryMode this property is used to start transactions with external card entry.

##### Prototype

```
eft.Track2 = ...
```

##### Data Format

The format must strictly follow this scheme:

```
; pan[12..19] = y[2] m[2] data[0..20] ?
```

##### Example

```
; 9756123100000004=0312?0
```

This property may be set by the application when card data was entered manually or by an external reader.

##### Remark

to be out of PCI Scope only technical cards beginning with **9** can be used

#### 4.4.3.26 AutoMode Property

##### AutoMode Property

With AutoMode, the automatic behaviour of the driver can be controlled. AutoMode queries periodically the status of the device and reports changes of [DeviceEventCode](#), [DeviceStatusCode](#) and [ActivationState](#) with the [OnDeviceEvent-Event](#).

AutoMode sets also the unique transaction behaviour for the different devices.

AutoMode can be set:

- as commandline parameter ( **/AutoMode** )
- AutoMode Property. This has priority over the commandline value. It has to be set before the first **Open**, **Close** or **QueryStatus**

Possible Values:

**EFT\_AMODE\_NULL** (commandline: **null** value: 0 )  
all automatic behaviour is switched off.

**EFT\_AMODE\_DEFAULT** (value 1)  
as AutoMode Property: The values are taken from the commandline. If not set in the commandline: default is amodeTestConnection + amodeTestCardInIdle + amodeTestCardbeforeTrx + amodeAssumeDevicePresent + amodeAutoActivate

**EFT\_AMODE\_TEST\_CONNECTION** (commandline: **testconn** value: 2 ) default  
Test the connection to the device (polls the status of the device every 10sec in OpenIdle state). If there is no answer during 1 minute, a DeviceStatusCode deviceNoAnswer is reported.

**EFT\_AMODE\_TEST\_CARD\_IN\_IDLE** (commandline: **testcard** value: 4) default  
Tests, if a card is inserted in OpenIdle state (polls the status of the device every 2sec in OpenIdle) (for VEZ+ no polling is done, because this device reports cards automatically). The insertion of a card is recognized and reported with the DeviceEventCode.

**EFT\_AMODE\_TEST\_CARD\_BEFORE\_TRX** (commandline: **cardbefore** value: 8 ) default  
in a transaction, the command is sent to the Device after a card is inserted (and, for some cards, a correct PIN is entered). For this, it polls the status of the device every 1sec.  
exceptions:  

- o for transactions, which need no cards, the command is sent immediately to the device
- o for VEZ+, the transaction is sent immediately because VEZ+ allows an Abort.
- o for KESS, the transaction is sent immediately, if the status is 2 (offen) or 4 (Kunde bereit), during PIN-entry (status 3), the Trx is kept until status == 4.

**EFT\_AMODE\_FORCE\_CARD\_BEFORE\_TRX** (commandline: **forcecardbefore** value: 16 )  
like amodeTestCardbeforeTrx but also for VEZ+ and KESS

**EFT\_AMODE\_VEZP\_NO\_COUNTER\_PICKUP** (commandline: **veznocounter** value: 32 )  
no counter pickup is done during open (VEZ+) . see also [Applications Collection](#)

**EFT\_AMODE\_BALANCE\_FROM\_DRIVER** (commandline: **balancefromdriver** value: 64 )  
instead of Device, the BalanceCounters are calculated by the driver. see also [Applications Collection](#)

**EFT\_AMODE\_ASSUME\_DEVICE\_PRESENT** (commandline: **assumedevice** value: 128 ) default  
after the start of the driver, a device is searched by the driver. In this time, the driver assumes a device and answers the commands Open, Close and QueryStatus with DeviceStatusCode EFT\_DEVICE\_NOT\_INIT. When a device is found, the last Open or Close command is executed and the actual DeviceStatusCode is reported with a status message (OnDeviceEvent). So the DeviceStatusCode can be monitored from beginning.

**EFT\_AMODE\_AUTO\_ACTIVATE** (commandline: **autoact** value: 256 default )  
if the Terminal cannot be opened or closed, the driver does it itself, when the terminal can be accessed again. the status change is reported then.

**EFT\_AMODE\_AUTOMAT** (commandline: **automat** value: 512 )  
the behaviour of the driver is optimized for vending machines. TrxTimeout and Change of Amount are possible. The EFTSimulator has behaviour like a machine module.

**EFT\_AMODE\_AUTOMAT\_AcceptCardFirst** (value: 1024)

**EFT\_AMODE\_AUTOMAT\_ReadBrands** (value: 2048)

**EFT\_AMODE\_AUTOMAT\_ReadBrandsInTrx** (value: 4096)

**EFT\_AMODE\_AUTOMAT\_AllowStopOnTrm** (value: 8192)

**EFT\_AMODE\_AUTOMAT\_WithContactLess** (value: 16384)

**EFT\_AMODE\_NO\_AUTO\_LOGIN** (value: 32768)

#### 4.4.3.27 PrintCharSet Property

##### PrintCharSet Property

the character set to be used for Printing

Optional the default is 1 (Latin-1)

Number	Character Set	Type
0	UTF-8	
ISO 8859:		
1	Latin-1	Western European
2	Latin-2	Central European
5	Latin	Cyrillic
15	Latin-9	Western European

#### 4.4.3.28 PrinterState Property

##### PrinterState Property

shows printer status after calling the print method

EFT_PRINTER_READY	printer is ready for the next print request
EFT_PRINTER_INVALID_STATE	invalid state to print
EFT_PRINTER_INVALID_CHARACTER	invalid character in PrintText
EFT_PRINTER_BUSY	printer is busy
EFT_PRINTER_OUT_OF_PAPER	the printer is out of paper
EFT_PRINTER_TIMEOUT_ECR	the cash did not supply data to the open job for longer time
EFT_PRINTER_MAX_SIZE_EXCEEDED	PrintText is to large (EFT internal buffer overflow)
EFT_PRINTER_NO_PRINTER	EFT is not equipped with a printer or EFT doesn't permit to use the printer

Defined in enum EFTPrinterState

#### 4.4.3.29 TipAmount Property

##### TipAmount Property

Returns of the given the amount of tip

*Remark: newTIP since ep2 V4.1. The old TIP mode will be indicated with the Value -1*

#### 4.4.3.30 CustomerLanguage Property

##### CustomerLanguage Property

Returns the customer respectively cardholder language. The definition in accordance with ISO 639 language codes

#### 4.4.3.31 DeviceId Property

##### DeviceId Property

DeviceId to be used. Default is the first available.

Using a terminal connected to the Ethernet the DeviceId is to set with the ep2:TrmID. This allows the use of a simple and universal MPD configuration.

#### 4.4.3.32 pay@theTable Properties

##### pay@theTable Properties

not supported anymore

#### 4.4.3.33 NbrInstalments Property

##### NbrInstalments Property

Returns the Number of Instalments for this authorization.

#### 4.4.3.34 SuppressReceiptOnTerminal Property

##### SuppressReceiptOnTerminal Property

If the terminal is configured that it can print the receipt independently with this property it is possible to suppress the autonomous print activity

#### 4.4.3.35 Petrol Properties

##### Petrol Properties

###### HasCardInfos

0	no additional infos available	
> 1	bitmap indicates available informations	
1	EFT_HAS_CARD_INFO	additional infos available
2	EFT_HAS_CARD_PAN_DATA	magstripe PAN data available
4	EFT_CHIP_DATA_READING	a chip was recognized and an event will follow when ChipData will be available
8	EFT_HAS_CHIP_PAN_DATA	ChipData are available
16	EFT_CARD_SWIPED	the card was read by pulling out

the following properties (values) are available, if the card is loaded (HasCardInfos = 1)

ProcessingTarget:	0=EFT, 1=ECR
ContextId	
ApplicationName	ApplicationLabel
CardTrack2	
CardTrack3	
CardIcc	
CardPanHash	
ChipPanHash	
CardNumber	CardPAN
CardAppExpDate	
CustomerLanguage	language out the authorization
CardLanguage	language code of the card

**ProceedNeeded Property:**

These properties are used to handle petrol transactions see also [Petrol Transaction Flow](#).

The ProceedNeeded Property has to be set before the Transaction to:

EFT_PutProceedNeeded	"0"	normal Transaction without ProceedNeed (default)
	"1"	start transaction with ProceedNeeded
	"2"	Proceed after receiving card data

**Remarks**

- The use of ProceedNeeded option is limited to the transaction start before card information are available.
- The main reason to use this option is to give the ECR a possibility to fill the [Petrol Sales Item](#) after the card was already inserted and the transaction is ongoing.
- In case the option ProceedNeeded is set, and the MOC card is swiped, the terminal will perform a so called Silent Abort of the EFT transaction and wait for the further instructions of ECR.
- If the option ProceedNeeded is set at transaction start, the terminal will expect an additional message to confirm the transaction.
- When the ECR is done with the card it should send the [Card Eject Event](#).
- Further information can be found in the "EFT Terminal Framework / IFSF EPS Interface" specification.

**4.4.3.36 OperationPhase Property****OperationPhase, Property**

The property contain information about the status of a petrol transaction.

**Property**

OperationPhase                      integer                      see table below

**OperationPhase Property:**

0	no transaction
1	Processing
2	Prepared to commit
3	Committed
4	Rolled back

**4.4.3.37 AmountOther Property****AmountOther Property**

Additional Amount added to the transaction amount

*Remark:* since ep2 V5.3

**4.4.3.38 FeeAmount Property****FeeAmount Property**

Additional FeeAmount for example in case of cashback

*Remark:*

**4.4.3.39 AcqID Property****AcqID Property**

ep2 Acquirer Identifier

*Remark:* the left most (first) digit is 0

**4.4.3.40 StatKeyPANRctInd Property****StaticKeyPanIndex Property**

two digits number shown on the merchant receipt

*Remark:* ep2 Static Key PAN Receipt Index. Identifies the <Static Key PAN Receipt> in conjunction with the <Acquirer Identifier>

#### 4.4.3.41 SubBrand Property

##### SubBrand Property

ep2 SubBrand. used for the distinction of sub-brands

*Remark:*

#### 4.4.3.42 Application Label Property

##### ApplicationLabel Property

Label shown on authorization receipts

*Remark:* Mnemonic associated with the <Application Identifier (AID)> according to [ISO7816], part 5.

#### 4.4.3.43 AccountIndex Property

##### AccountIndex Property

numeric value between 0 and 9, which define the account which should be addressed

*Remark:* not all terminals support this functionality. further information is available from the terminal supplier

#### 4.4.3.44 ECRSeqCounter Property

##### ECRSeqCounter Property

numeric value between 1 and 99'999'999,

Counter maintained by the Cash Register, which is incremented by one for each sale transaction. A sale transaction may contain 1..n eft transactions.

*Remark:* not all terminals support this functionality. further information is available from the terminal supplier

#### 4.4.3.45 LoyaltyMode Properties

##### LoyaltyMode Properties

These properties are used to handle the Loyalty transaction - see also [LoyaltyTransactionOperation](#)

##### LoyaltyMode Property:

- 0 EFT\_LOYALTY\_NO\_LOYALTY
- 1 EFT\_LOYALTY\_INIT
- 2 EFT\_LOYALTY\_INFO
- 3 EFT\_LOYALTY\_CONTINUE
- 4 EFT\_LOYALTY\_AUTHORISATION\_INFO
- 5 EFT\_LOYALTY\_AUTHORISATION\_NOINFO

The LoyaltyMode Property has to be set before the Transaction to:

EFT_LOYALTY_NO_LOYALTY	normal Transaction without Loyalty (default)
EFT_LOYALTY_INIT	start a Loyalty phase 1
EFT_LOYALTY_CONTINUE	continue an authorization phase after a Loyalty phase 1 ok answer

to start a Loyalty transaction in one phase set the LoyaltyMode Property to EFT\_LOYALTY\_CONTINUE

The LoyaltyMode Property is set to the following value after a successfull Transaction request:

EFT_LOYALTY_NO_LOYALTY	normal Transaction without Loyalty (default)
EFT_LOYALTY_INFO	Loyalty phase 1 ready, Loyalty information available
EFT_LOYALTY_AUTHORISATION_INFO	the transaction was successfull Loyalty information is available
EFT_LOYALTY_AUTHORISATION_NOINFO	the transaction was successfull no Loyalty information is available

##### Remarks

- some EFT Terminals aren't able to start with EFT\_LOYALTY\_INIT (eg. IFSF eps) in this case the transaction has to be done in one phase LoyaltyMode Property = EFT\_LOYALTY\_CONTINUE
- a Loyalty transaction can be combined with a DCC transaction when doing one phase Loyalty (setting LoyaltyMode = EFT\_LOYALTY\_CONTINUE and DccMode = EFT\_DCC\_CONTINUE\_DCC)



#### 4.4.3.46 LoyaltyData Property

##### LoyaltyData Property

Contains proprietary Loyalty information available depending the LoyaltyMode - see also [LoyaltyTransactionOperation](#)

#### 4.4.3.47 LoyaltyInfo Property

##### LoyaltyInfo Property

Contains proprietary Loyalty information sent from the Cash Register to the EFT - see also [LoyaltyTransactionOperation](#)

### 4.4.4 Events

#### 4.4.4.1 OnStatusChange Event

##### OnStatusChange Event

The runtime library raises this event, when the state of the session changes.

```
OnStatusChange ( StatusCode )
```

##### Parameters

*StatusCode*

See [Status](#) property for a list of possible status values.

#### 4.4.4.2 OnCommandComplete Event

##### OnCommandComplete Event

The driver raises this event, when a pending operation is ready to complete on the session in async mode. The application must call the [Complete](#) method after receiving this event to finish the operation.

```
OnCommandComplete ( exceptionCode )
```

The *exceptionCode* parameter contains the result code of the operation that has completed.

The property [CompletedCommand](#) shows, which Command has completed. this must be read before the Call of Complete.

#### 4.4.4.3 OnError Event

##### OnError Event

##### Summary

This event is called if an error occurs during request-processing.

```
OnError ( exceptionCode, message );
```

##### See also

[ExceptionCode Property](#)

#### 4.4.4.4 OnDeviceEvent Event

##### OnDeviceEvent Event

##### Summary

This event is fired by the device whenever an event of interest to the ECR occurs (change of [ActivationState](#) , [DeviceStatusCode](#) or [DeviceEventCode](#) )

```
OnDeviceEvent ( deviceEventCode [ , applicationKey ] )
```

##### Arguments

*deviceEventCode*

Specifies the type of event:

EFT\_READER\_EMPTY  
EFT\_READER\_LOADED

The card has been removed from the reader

A payment card has been read or inserted and a new transaction may be started within a device specific timeout. The optional applicationKey argument indicates which application will handle the transaction with the inserted card.

**EFT\_READER\_EJECTED** The card has been ejected from the reader but has not yet been removed from the slot.  
This event is not generated for manual readers.

Defined in enum `EFTDeviceReaderStatus`

#### *applicationKey*

Indicates the application which will handle the card. Used in conjunction with the `EFT_DEV_CARD_PRESENTED` event.

#### **See Also**

[Applications Property](#)

#### **Restrictions**

Some events are limited to particular device types.

### 4.4.4.5 OnDeviceMsg Event

#### **OnDeviceMsg Event**

##### **Summary**

This event is fired by the device whenever the Terminal has a message to send to the ECR

```
OnDeviceMsg ( deviceMsgCode, deviceMsg)
```

#### **Arguments**

##### *deviceMsgCode*

Specifies the type of message:

`EFT_DC_Display`  
`EFT_DC_ClearDisplay`  
`EFT_DC_PARGetTotalAmount`  
  
`EFT_DC_PARTransactionRequest`  
`EFT_DC_PARCashTransaction`  
`EFT_DC_PARAbortFlow`  
  
`EFT_DC_PARPrintRequest`  
`EFT_DC_PARGetReceipt`  
  
`EFT_DC_PARGetReceiptPaid`  
  
`EFT_DC_PARActorVerification`  
  
`EFT_DC_PARActorPwdeCRVerification`  
  
`EFT_DC_PARActorPwdeFTVerification`  
  
  
`EFT_DC_KeyPressed`

The ECR should display the Text in deviceMsg  
The ECR should clear the displayed text.  
The Terminal requests a pay@theTable Get Total Amount.  
The Terminal requests a pay@theTable Transaction.  
The Terminal signals a pay@theTable Cash Transaction  
The Terminal signals to interrupt a pay@theTable Transaction flow  
The Terminal signals to get the ECR Receipt  
The Terminal signals a pay@theTable ECR Table Receipt  
The Terminal signals a pay@theTable ECR Receipt Paid Amount  
The Terminal signals a pay@theTable Actor verification to be done by ECR  
The Terminal signals a pay@theTable Actor and Password verification to be done by ECR  
The Terminal signals a pay@theTable Actor verification to be done by ECR with following Password verification by the EFT  
The Terminal signals a key activity for IFSF EFT Terminals

Defined in enum `EFTDeviceMsgCode`

##### *deviceMsg*

a string depending on the `deviceMsgCode`.

#### **See Also**

[pay@theTable](#)  
[Petrol Functions](#)

## 4.5 Applications Collection

### 4.5.1 Applications Collection

#### **Applications Collection**

Represents a collection of Application objects.

#### **Properties**

**Count** Returns count of items in the collection

**Item** Returns an item specified by a key or zero based index

Applications holds the possible applications (card products) of the device. Each possible currency has an own application. The applications are read from the terminal after a balance request and are present after a connect from the latest balance. During open, the applications are read for VEZ+ (see also **amodeVEZpNoCounterPickup** ) and KESS.

## 4.6 Application Object

### 4.6.1 Application Object

#### Application Object

Represents an EFT application card-processing application information object. Holds also the information of the latest balance reply.

#### Properties

All properties are read-only.

<b>Key</b>	Opaque, unique identifier for the application
<b>Name</b>	Display name, e.g. "American Express"
<b>AID</b>	[EP2] Application Identifier
<b><a href="#">BrandId</a></b>	unique Brand identifier
<b>TerminalId</b>	Returns a string containing the identification number the application uses for the device.
<b>Currency</b>	Returns currency accepted by this application.
<b><a href="#">ApplicationType</a></b>	
<b>PeriodNumber</b>	Of the latest balance reply
<b>TotalCount</b>	Of the latest balance reply
<b>TotalAmount</b>	Of the latest balance reply
<b>DebitCount</b>	Of the latest balance reply
<b>DebitAmount</b>	Of the latest balance reply
<b>CreditCount</b>	Of the latest balance reply
<b>CreditAmount</b>	Of the latest balance reply
<b>VoidCount</b>	Of the latest balance reply
<b>VoidAmount</b>	Of the latest balance reply
<b>ReservationCount</b>	Of the latest balance reply
<b>ReservationAmount</b>	Of the latest balance reply

### 4.6.2 Application Type

#### ApplicationType Property

this property of the Application Object shows the type bitwise OR assembled:

Value	Description
1	this Application cannot be rolled back (e.g. CASH card) ( such an application must also be committed. In case of <i>Commit(false)</i> , <i>Outcome</i> will be <i>true</i> )

## 4.7 Receipt Object

### 4.7.1 Receipt Object

#### Receipt Object

Encapsulates receipt data and preformatted text.

```
receipt.ApplicationKey
receipt.Value ( [ fieldName ] )
```

The Value property is read-only and can be used to retrieve individual receipt fields or the preformatted text. The ApplicationKey property returns the key of the application that was used for the transaction.

#### Parameters

*fieldName*

Optional name of requested receipt field (see [transaction-auth Message](#)). Receipts are generally protocol and application specific. If an empty string is supplied (the default value), preformatted data is returned.

#### Return Value

Read access to Value property will always succeed, even if an invalid *fieldName* is supplied. An empty string will be returned in this case.

**Data Format**

If preformatted receipt data is requested, the property value is a string containing fixed width lines separated by carriage return new-line characters (ASCII 0x13 0x10). Two subsequent new-line characters indicate a paper-cut operation. Field values are returned as strings.

**Example**

Example of balance receipt containing C-style format characters

```

KASSIERER ( IN)                0000\r\n
TERMINAL/KASSE                 5677\r\n
BELEG-NR.                     004150\r\n
                                \r\n
TAGESABSCHLUSS                \r\n
                                \r\n
American Express              \r\n
TOTAL          CHF      4      0.00\r\n
BUCHUNG        CHF      1      1.30\r\n
GUTSCHRIFT     CHF      1      1.30\r\n
STORNO         CHF      2      0.00\r\n
                                \r\n
TOTAL          CHF      4      0.00\r\n
\r\n\r\n

```

## 4.8 Devices Collection

### 4.8.1 Devices Collection

**Devices Collection**

Represents a collection of Device objects.

**Properties**

<b>Count</b>	Returns count of items in the collection
<b>Item</b>	Returns an item specified by a key or zero based index

## 4.9 Device Object

### 4.9.1 Device Object

**Device Object**

Represents information about a device available through the driver.

**Properties**

<b>Deviceld</b>	Identifier for the device. May be used to select the device in subsequent Open calls.
<b>Port</b>	Name of the serial port device used to communicate with the device.
<b>Class</b>	Class of driver used to communicate.
<b>ProductName</b>	Optional driver/EFT software version.
<b>SoftwareVersion</b>	Optional driver/EFT software version.

**Retrieving the Device List**

The following example code shows how to select a particular device from the device list at startup (with no error checking):

```

eft.Connect
eft.DeviceControl(32, 1)
eft.DeviceId = eft.Devices(0).DeviceId
eft.Open

```

## 5 Integration

### 5.1 Integration under Windows

#### 5.1.1 Windows NT Systems (NT4, 2000, XP, 2003, Vista, 2008, 7 and Windows 8

##### 5.1.1.1 Using under Windows NT Systems

###### Windows NT Systems

The driver can be [configured](#) to run as a service under Windows NT Systems. This option will not be set during installation.

##### 5.1.1.2 Visual Basic Script Example

###### Visual Basic Script Example

The runtime library can be accessed from the Windows Scripting Host using synchronous method calls. To create a new EFTDriver object

```
Set eft = CreateObject("eftoa.EFTDriver")
```

is called. Once this is done, we can configure and open the EFT device by calling

```
eft.Async = FALSE
eft.ECrid = "1";

eft.Connect
eft.Open
```

A runtime error will be generated, when the function fails. A new transaction can be started now:

```
eft.Currency = "CHF"
eft.Transaction "debit", 100
```

If no error occurs, the device is ready to

```
eft.Commit TRUE
```

and the transaction has completed. After that we may

```
eft.Close
```

finish the shift. Other methods and properties work as well from WSH.

#### 5.1.2 Windows CE

##### 5.1.2.1 The Windows CE Binaries

###### The Windows CE Binaries

The Windows CE Binaries can be found under the *wince* directory of the distribution. The application must start the driver process manually (because Services are not supported under CE) by executing the **eftdvs.exe** file. See [Configuration](#) for more details.

note that com ports are marked with double point in Windows CE ( e.g. "COM1:")

## 5.2 Integration under Linux

### 5.2.1 Integration under Linux

#### Integration under Linux

Under Linux and other variations of unix, only the Java and C-API interfaces are supported. The **eftapi.so** shared library

module must be placed in the proper location, such as /usr/lib.

## 5.3 Integration under OSX

### 5.3.1 Integration under OSX

#### Integration under OSX

Under OSX only the Java and C-API interfaces are supported. The **libeftapi.dylib** dynamic library module must be placed in the proper location.

## 6 Using COM

### 6.1 Using the COM/OLE Interface

#### Using the COM/OLE Interface

Under Windows, the EFT driver exposes an automation interface that can easily be accessed by Automation Containers, such as VBA, Excel and native applications under C++. The Type-Library is named "Telekurs EFT/MPD Library". It is embedded in the eftoa.dll binary.

## 7 Using Java

### 7.1 Using the Java interface

#### Using the Java interface

The Java classes which implement the runtime part that plugs into the ECR application are packaged into the **eftoa.jar** file. Make sure that the file is found by your Java Virtual Machine. This implementation requires JRE 1.5 or higher. Properties are accessed by Java-like accessor functions using the set/get prefixes.

A very simple implementation, that creates a driver object looks like this (error handling omitted):

```
import tk.eft.Driver;

class ECRSample
{
    public static void main(String [] args)
    {
        Driver driver = new Driver();

        driver.setCurrency("CHF");
        driver.setECRId("1");
        driver.Connect();
        driver.Open();
        driver.Transaction("debit", 100, 0);
        driver.Close();
    }
}
```

**Events:** to work with the Events, the processing object has to implement tk.eft.DriverEventsInterface. to enable the Events driver.AttachEventMonitor(this) has to be called.

#### Remarks:

Examples for the sync/async modes (see [Async Property](#)) and transaction program flow (see [Transaction Programming Flow](#)) exists:

**MPDJAVA\_Test** shows:

- Transaction sync
- Transaction async, polling timed with *Complete(timeout)*
- Transaction async, waiting for change with *Complete(-1)*
- Transaction in a Thread, async, waiting for change with *Complete(-1)*

**MPDJAVA\_Test\_1** shows:

- Transaction sync, watching changes with OnDeviceEvent
- Transaction sync in a thread, watching changes with OnDeviceEvent

**MPDJAVA\_Test\_2** shows:

- watching changes with `OnDeviceEvent`
- waiting for completion of a async command with `OnCommandComplete`
- commands sync and async
- handling events (`OnCommandComplete`) to be executed asynchronously on the AWT event dispatching thread

## 7.2 Using the Events in Java

### Using the Events in Java

the Events `OnStatusChange`, `OnError`, `OnCommandComplete` and `OnDeviceEvent` can be used, to monitor the process of the driver, especially in async mode.

for this the processing object has to implement `tk.eft.DriverEventsInterface` and call `driver.AttachEventMonitor(this)`

#### Remark:

the Events are fired from a different thread than the event dispatching thread of the GUI. To handle these Events with Swing, see

<http://java.sun.com/products/jfc/tsc/articles/threads/threads1.html>

how to handle this with swing: see also **MPDJAVA\_Test\_2**

#### Example:

```
public class MPD_Test_GUI extends javax.swing.JDialog implements
tk.eft.DriverEventsInterface{

    tk.eft.Driver driver;
    ..
    ..
    class OnCommandCompleteRunnable implements Runnable {
        int exceptionCode;
        public OnCommandCompleteRunnable(int exceptionCodeR)
        {
            exceptionCode = exceptionCodeR;
        }
        public void run()
        {
            /* execution of the OnCommandComplete */
            /* to be executed asynchronously on the AWT event dispatching thread */
            int completedCommand = driver.getCompletedCommand();
            driver.Complete(0);
            ..
            ..
        }
    };

    public void OnCommandComplete(int exceptionCode) {
        OnCommandCompleteRunnable doOnCommandComplete
            = new OnCommandCompleteRunnable(exceptionCode);
        javax.swing.SwingUtilities.invokeLater(doOnCommandComplete);
    }
    ..
    ..
    public MPD_Test_GUI(java.awt.Frame parent, boolean modal) {
        ..
        driver = new tk.eft.Driver();
        ..
        driver.AttachEventMonitor(this);
        ..
    }

};
```

## 7.3 Java ECR Simulator

### Java ECR Simulator

The sources or

Minimal requirements: Installed Java Runtime Environment version 1.4 or higher

The *ECR Simulator* is a simple Simulator of Electronic Cashier Register (ECR) written in Java. It supports the basic functionality of EFT devices like Open and Close operations, Transactions and Balances as well as configuration of specific settings.

The ECR Simulator is based on the Java Interface (eftoa.jar) of the Multiprotocol Driver. Before starting the simulator, make sure that the EFT Driver is running and the ECR Device(s) are detected successfully.

The Application consists of four individual windows. Each Window can be shown and hidden individually. The general purposes of the windows are listed below:

**Main Window:** provides the basic functions between ECR and MPD.

Please Note:

- The amount for transactions is given in centimes
- To perform a Transaction with passed Track2 Data, the format must follow the scheme:  
**;PAN=YMD?**  
PAN: credit card number (12 to 19 digits)  
Y year of expiration (2 digits)  
M month of expiration (2 digits)  
D additional data (0 to 20 digits)

**ReceiptLog Window:** Lists all generated Receipts received from the MPD.

**StatusLog Window:** Lists result codes of MPD method calls as well as several events thrown by the MPD Java Interface

**Property Window:** This Window lists all EFT and ECR properties supported by the Java Interface.

Please Note:

- "Application List" is only loaded after a balance operation
- The "Device List" is loaded after the device control operation "Driver Query DeviceList"

Not all EFT Terminals support every available device control commands

## 8 Using .NET

### 8.1 Using the .NET interface

#### Using the NET interface

The NET interface is implemented in the **tk\_eft.dll** component.

Add to your project the project->link->NET tk\_eft.dll

A very simple implementation, that creates a driver object looks like this (error handling omitted):

```
// Declaration
tk_eft.Driver eft;

// Construction
eft = new tk_eft.Driver();

// Execution
eft.ECrid = "1";
eft.Connect();
eft.Open();
eft.Currency = "CHF";
eft.Transaction("debit", 500, 0);
eft.Commit(true);
eft.Close();

// CleanUp
eft.Dispose();
```

**Events:** to work with the Events, the processing object has to implement Handlers and set them to the driver (see also **MPDNETGUI\_Test\_1** example)

```
eft.OnCommandCompleteEvent +=
    new tk_eft.OnCommandCompleteDelegate(this.OnCommandCompleteHandler);
eft.OnDeviceEventEvent +=
    new tk_eft.OnDeviceEventDelegate(this.OnDeviceEventHandler);
eft.OnStatusChangeEvent +=
    new tk_eft.OnStatusChangeDelegate(this.OnStatusEventHandler);
```



events should be marshaled into the event handling thread of the calling object when they affect graphic objects. This is done by the method **SetEventsMarshaled**

```
// Set Events OnDeviceEvent and OnCommandComplete
// to be marshaled into this event handling thread
eft.SetEventsMarshaled(this, tk_eft.Driver.EFT_ME_OnDeviceEvent |
    tk_eft.Driver.EFT_ME_OnCommandComplete);
```

#### Remarks:

Examples for the sync/async modes (see [Async Property](#)) and transaction program flow (see [Transaction Programming Flow](#)) exists:

**MPDNETGUI\_Test\_1** shows:

- Transaction sync
- Transaction async, waiting for completion with OnCommandComplete
- Transaction sync in a Thread
- Handling of Events
- Marshaling of events into the event handling thread

## 9 Using EFTAPI

### 9.1 EFTAPI overview

#### EFTAPI Overview

In the C-interface all methods and properties take an additional parameter containing a handle to the EFT device object. Definitions can be found in the **eftapi.h** file, the **eftapi.lib** must be linked with the client application.

Methods are Called with

```
EFT_MethodName(heft, ...);
```

Properties are read with

```
EFT_GetPropertyName(heft, &value);
```

Properties are set with

```
EFT_PutPropertyName(heft, newvalue);
```

#### Example

```
#include "eftapi.h"

int main()
{
    EFT_HANDLE heft;
    EFT_CreateSession(&heft);
    EFT_PutECRId(heft, "9999");

    EFT_Connect(heft);
    EFT_Open(heft);
}
```

#### Remarks:

the status fields ( [ActivationState](#) , [DeviceStatusCode](#) or [DeviceEventCode](#) ) are only updated during a EFT\_Complete call, or during a Command in synchron mode. So to read these fields in in EFT\_S\_IDLE, EFT\_S\_CONNECTED and EFT\_S\_PREPARED state call:

```
r = EFT_Complete(heft, 1); // see also Complete Method
```

before reading the status fields. See example EFTAPI\_Test

EFTAPI does **not** fire any events

for asynchron mode:

```
r = EFT_Complete(heft, -1); // see also Complete Method
```

can be used, to wait for an event, after return either [CompletedDeviceEvent](#) or [CompletedCommand](#) is set. this can be done in a separate thread. See example EFTAPI\_Test\_2

see how to access [Applications](#) in EFTAPI\_Test example

**Examples** for the sync/async modes (see [Async Property](#)) and transaction program flow (see [Transaction Programming Flow](#)) exists:

**EFTAPI\_Test** shows:

- Transaction sync
- Transaction async, polling timed with *Complete(timeout)*
- Transaction in a Thread, async, waiting for change with *Complete(-1)*

**EFTAPI\_Test\_2** shows:

- a thread for waiting for changes of ReaderStatus and completion of commands with *Complete(-1)*
- commands async, waiting for changes in the CompleteThread

## 9.2 EFTAPI Applications

### Applications:

in EFTAPI, the applications can be reached with the following functions:

```
int EFT_GetApplicationsCount(heft, &nCount)
```

gives the number of applications in *nCount*

```
int EFT_GetApplicationsProperty(heft, i, PropertyName, value, sizeof(value))
```

gives the value of the property in value as string from application i.

PropertyNames:

```
"application-key"
"application-name"
"aid"
"brand-id"
"currency"
"terminal-id"
"application-type"
```

the following Properties are only set on balanced applications:

```
"period-number"
"total-count"
"total-amount"
"debit-count"
"debit-amount"
"credit-count"
"credit-amount"
"void-count"
"void-amount"
"reservation-count"
"reservation-amount"
```

see example EFTAPI\_Test

## 10 Configuration and Debugging

### 10.1 Configuration and Debugging

#### Configuration and Debugging

The driver is configured through the command line at startup. The following command line options are defined:

#### /Service

Installs the program as a Windows Service.

Switches in combination with /Service:

**/AutoStart** Install the Service to be started on system startup automatically (default: Manual Starting)

**/Username uname**

**/Password pw** Account to run the service, default = System-account

use the Configuration File (**eftdvs.cfg**) to set the running options of the service

use NET START EFTMPD to start the service

use NET STOP EFTMPD to stop the service

if the Service is not running, when the client (COM or EFTAPI) tries to connect the first time, then the client starts

the service.

In this case, the client stops the service on termination.

**/UnRegServer**

Removes the Service

**/Console**

Runs the program on console. This parameter must be supplied to run the driver on console.

**/AutoMode <options>**

see [AutoMode Overview](#)

**/Trace <options>**

Sets [trace and debugging](#) options. Supply -1 to enable all output.

**/TraceDir <path>**

Sets the directory for the trace output used for logging See [trace and debugging](#)

**/TraceOutput <path>**

Sets the path of the trace output file. See [trace and debugging](#)

**/OPI**

start MPD in O.P.I. mode

**/OPI <mode>**

With mode the protocol compatibility can be set using a combination of the following values.

Possible Values:

**OPI\_V1.3** (value 1) default  
O.P.I. V1.3

**OPI\_V1.2** (value 2)  
O.P.I. V1.2

**OPI\_V1.3\_CCREDIT\_ALL** (value 4)  
Authorization Response with ccredit:RctDetails

**OPI\_WAIT\_CLIENT\_ACK** (value 8)  
For Vending Machines, when a Commit is required

**OPI\_NAME\_SPACES** (value 16)  
Inserts the XML namespaces in the response

**OPI\_PERSISTENT\_PRINTER\_CONNECTION** (value 32)  
Keeps the connection to the OPI device persistent

**OPI\_DISABLE\_LOGIN\_LOGOFF\_RECEIPTS** (value 64)  
Disable the printing of login and logoff receipts

**OPI\_DISABLE\_RECONCILIATION\_WITH\_CLOSURE\_RECEIPTS** (value 128)  
Disable the printing of reconciliation receipts

**/OPISchemaLocation <path>**

set the path where the schemas are located, if needed

**/OPIServerPort <port>**

set the server port where mpd is listening, default is 20002

**/OPIDevicePort <port>**

set the device port (eg display printer), default is 20007

**/TextNoRef**

text in messages has no references for special characters ('ä' -> 'ae')

**/NoEvents**

MPD sever answers only on Requests. It does not send Status or Exception messages without request.

**/Port <port-name>**

Configures the driver to use specified communications port. /Port 0 disables serial communication (e.g. only VEZ +overIP)

e.g. Win32: /Port COM1  
e.g. Linux: /Port /dev/ttyS0

**/Speed <speed>**

Drivers uses only the given speed

**/Protocol <protocol>**

Drivers uses only the given protocol

VEZ	VEZ 5.1
VEZP	VEZ plus (ep2)
KESS	
EP2	eps xml
IFSFS [IP-Address]	IFSFS eps IP Connection [EFT Address eg. 10.11.12.13]
IFSFSer [Port Speed]	IFSFS eps Serial Connection [EFT Port eg. \\.\COM1 115200]

**/NoProbe**

normally the driver does a probing during initialization. with /NoProbe this is not done. /NoProbe needs /Port, /Speed, and /Protocol to be set

**/AutoStart**

normally the device driver is started on the first connect. with /AutoStart set the device driver is started immediately after initialization.

**/VEZMode <options>**

Configure the driver to reZ-extension functions from the EFT device. Possible values are:

Option	Value	Description
	1	Enables the use of VEZ extension for the DaVinci-Terminal
	4	Enable EPSYS Extension (Init function). It is available through <a href="#">DeviceControl</a> .

To start a troubleshooting session, perform the following steps:

1. Stop the *EFT Multiprotocol Service* if installed and running.
2. Open a command shell (cmd.exe) and switch to the installation directory.
3. On the command line, enter

```
eftdvs /Console /Trace -1
```

The driver will write all debug message to the console output.

**/EFTPort <port>**

The port where MPD is waiting for TRM connecting for VEZ+overIP. default is 8138.

**/ECRPort <port>**

The port where MPD is waiting for connection from ECR. default is 8137.

**/ForceDCC**

this parameter force a DCC transaction (equal DCC-mode ="3")

**/TID <tid>**

the default TID to be searched in LAN. Value 0 all unbounded TID's will be called in LAN.

**/DisableBroadcast**

do not send Broadcast messages to search for device in LAN

**/MPDAddress**

the Broadcast Message will contain this address to indicate to the terminal to which network adapter the terminal has to connect to

**/NoSerialPortScan**

do not search for device on serial ports

**/NoZip**

do not compress the trace files

**/KESSGate**

use the Gateway for KESS (STP active)

**/DeleteLogAge**

this parameter can be used to define the log file storage time in month. Default value is 12 month.

**/KeepAliveMsg**

this parameter is to be set for generating a regular message flow from the server to the client

**/ReceiptOptions**

this parameter overrides the Property ReceiptOptions and has to be set with a decimal value

**Configuration File**

You can also create a configuration file **eftdvs.cfg** and put the command line options into it. Just keep them together on the first line of the file.

The file **eftdvs.cfg** has to be in the same directory as the eftdvs executable file.

You can also create a configuration file **eftdvscfg.xml** and put the options into it. The file **eftdvscfg.xml** has to be in the same directory as the eftdvs executable file.

see [Configuration with XML](#)

**/ConfigDir <path>** on the commandline

Sets the directory to search for the configuration file instead of the loadir

**Debugging with Log in MPD Client**

Logging of Procedure calls, returns and behaviour can be logged in the MPD client (EFTAPI or ocx).

You can create a configuration file **eftclient.cfg** and put the command line options into it. Just keep them together on the first line of the file. The file **eftclient.cfg** has to be in the same directory as the application exe.

possible options are:

**/Trace <options>**

**/TraceDir <path>** (result: eftclientYYYYMMDD.log)

**/TraceOutput <path>**

**Example :**

/TraceDir C:\Log

## 10.2 Configuration with XML

**Configuration with XML**

You can create a configuration file **eftdvscfg.xml** and put the options into it. The file **eftdvscfg.xml** has to be in the same directory as the eftdvs executable file.

If the MPD is not running as Service, the Commandline-Parameter **/Console** has to be set. all other parameters are set in the XML.

element	Config	m	
attributes	Trace	o	Sets <a href="#">trace and debugging</a> options. Supply "-1" to enable all output
	TraceOutput	o	Sets the path of the trace output file. See <a href="#">trace and debugging</a>
	TraceDir	o	Sets the directory for the trace output used for logging See <a href="#">trace and debugging</a>
	NoProbe	o	normally the driver does a probing during initialization. with NoProbe="1" this is not done. NoProbe needs Port, Speed, and Protocol to be set. NoProbe can be set here for all devices
	AutoStart	o	normally the device driver is started on the first connect. With AutoStart="1" set the device driver is started immediately after initialization. AutoStart can be set here for all devices
	VEZMode	o	Configure the driver to request VEZ-extension functions from the EFT device. Possible values are
	AutoMode	o	see <a href="#">AutoMode Overview</a>
	TextNoRef	o	="1": text in messages has no references for special characters ('ä' -> 'ae')
	Simulator	o	="1": Start the EFT-Simulator
	OPISchemaLocation	o	set the path where the schemas are located, if needed
	OPI	o	="1": OPI_V1.3 (default) O.P.I. V1.3 ="2": OPI_V1.2 O.P.I. V1.2 ="4": OPI_V1.3_CCREDIT_ALL Authorization Response with ccredit:RctDetails ="8": OPI_WAIT_CLIENT_ACK For Vending Machines, when a Commit is required ="16": OPI_NAME_SPACES Inserts the XML namespaces in the response ="32": OPI_PERSISTENT_PRINTER_CONNECTION Keeps the connection to the OPI device persistent

			="64": OPI_DISABLE_LOGIN_LOGOFF_RECEIPTS Disable the printing of login and logoff receipts ="128": OPI_DISABLE_RECONCILIATION_WITH_CLOSURE_RECEIPTS Disable the printing of reconciliation receipts  Remark: it is possible to combine the values, respectively to add
	OPIServerPort	o	set the server port where mpd is listening, default is 20002
	OPIDevicePort	o	set the device port (eg display printer), default is 20007
	NoEvents	o	="1": MPD sever answers only on Requests. It does not send Status or Exception messages without request.
	ForceDCC	o	this parameter force a DCC transaction (equal DCC-mode ="3")
	EFTPort	o	The port where MPD is waiting for TRM connecting for VEZ+overIP. default is 8138.
	ECRPort	o	The port where MPD is waiting for connection from ECR. default is 8137.
	TID	o	the default TID to be searched in LAN. default 0 (all).
	DisableBroadcast	o	="1": do not send Broadcast messages to search for device in LAN
	MPDAddress	o	eg. ="10.0.0.2": the Broadcast Message will contain this address to indicate to the terminal to which network adapter the terminal has to connect to
	NoSerialPortScan	o	="1": do not search for device on serial ports
	KESSGate	o	="1": use the Gateway for KESS (STP active)
	KeepAliveMsg	o	="1": is to be set for generating a regular message flow from the server to the client
	DeleteLogAge	o	this parameter can be used to define the log file storage time in month. Default value is 12 month. eg. 06 for 6 month
	ReceiptOptions	o	this parameter overrides the Property ReceiptOptions and has be be set with a decimal value
element	Device	m	one element for each device
attributes	Id	o	Device-Id if omitted -> counts the position
	Port	o	Configures the driver to use specified communications port. e.g. Win32: Port = "COM1" e.g. Linux: /Port = "/dev/ttyS0" If omitted, the available ports are scanned (only Win32). for more than one device, this attribute is mandatory
	Speed	o	Driver uses only the given speed
	Protocol	o	Drives uses only the given protocol ="VEZ" VEZ 5.1 ="VEZP" VEZ plus (ep2) ="KESS" ="EP2" eps xml
	NoProbe	o	normally the driver does a probing during initialization. with NoProbe="1" this is not done. NoProbe needs Port, Speed, and Protocol to be set. NoProbe can be set here for the specific device
	AutoStart	o	normally the device driver is started on the first connect. With AutoStart="1" set the device driver is started immediately after initialization. AutoStart can be set here for the specific device
	TID	o	the TerminalID to be connected for the given Id, for VEZ+overIP
	ECRId	o	the ECR Id for identification of the correct Terminal(TID) for a given Id and a given ECR

m = mandatory, o = optional

Example:

```
<Config Trace="logstd" TraceDir="%TEMP%" >
  <Device Id = "1" Port="\\.\COM2" Speed="115200" Protocol="VEZP" NoProbe = "1" AutoStart="1"/>
  <Device Id = "2" Port="\\.\COM1" Speed="115200" Protocol="VEZP"/>
</Config>
```

Examples for Terminals on IP:

```
<Config Trace="logstd" TraceDir="%TEMP%" >
  <Device Id = "1" TID="20101114"/>
  <Device Id = "2" TID="20109999"/>
  <Device Id = "3" TID="20101111"/>
</Config>
```

(the DeviceId 2 selects the Terminal with ID 2010999)

```
<Config Trace="logstd session" TraceDir="C:\Trace" >
  <Device Id = "1" ECRId="1234" TID="20101113"/>
  <Device Id = "1" ECRId="1235" TID="20101114"/>
</Config>
```

(the DeviceId 1 on ECR 1234 selects the Terminal with ID 20101113)

## 10.3 Work with the EFTSimulator

### Working with the EFTSimulator

a EFTSimulator is available in the installation (simul.exe). this EFTSimulator can be used on a serial port or with VEZ+overIP.

the configuration of the EFTSimulator can be set with commandline switches, or is asked on Startup when not set on commandline.

#### command line switch **/Profile**

- 1       davinci profile
- 2       davinci vending profile

#### command line switch **/Settings**

- a       VEZ over TCP on local host
- b       VEZ over TCP on remote host (/MPDAddress has to be set)
- c       VEZ over TCP, wait for Broadcast from MPD (cannot be run on same computer as MPD, because Broadcast are send only to other computers)
- 1..8    VEZ on serial COM1..8 baudrate 115200, or define Baudrate with /Speed

#### command line switch **/MPDAddress**

IP Address of MPD

#### command line switch **/Speed**

Baudrate on serial port

#### command line switch **/Deviceld**

Terminal ID to be emulated.

simul.exe /Profile 2 /Settings 1 /Speed 38400

emulated a davinci vending terminal with Baudrate 38400 and is connected at COM1

simul.exe /Profile 2 /Settings c /Deviceld 1234

emulated a davinci vending terminal with TID 1234 witch wait for a Broadcast from the MPD and connects then with VEZ+overIP

## 11 Trace and Debug Options

### Trace and Debug Options

Several trace and debug options may be activated to track driver operation. The following values may be combined to control the trace output:

**/Trace** options:

Flag	Description
logstd	Standard log. Traces all relevant activity, suppresses trace in idle states, when no changes
Msg	Trace internal messages
Control	Trace controller messages
Port	Trace serial port traffic
Vez	Trace VEZ driver
Session	Trace EFT session
opi	Trace OPIDevice requests
Server	Trace socket server
Gateway	Trace gateway activity
Service	Service activity
Journal	Debug journal access
Kess	Trace KESS driver
Warning	Emit extended warnings
pci	creation of PCI audit log

**/TraceDir** directory:

e.g.    /TraceDir c:\Temp  
         /TraceDir "c:\Program Files\Telekurs Card Solutions AG\EFT Multiprotocol  
 Driver\Trace"  
         /TraceDir %TEMP%

The Trace is written into the following files into the selected directory:

- daily in efdvsYYYYMMDD.log (YYYYMMDD stands for e.g.. 20040902)
- the old tracefiles will be zipped daliy into efdvsYYYYMMDD.log.zip, per week, the is a new zip-file.
- the Zip-files are deleted after one year.

/TraceDir overwrites /TraceFile.

if /TraceDir is set, and /Trace is not set, then /Trace logstd is set automatically.

It is highly recommended, to set /TraceDir permanently to have a log of the activity in case of problems.

**/TraceOutput filename:**

Sets the directory and filename for the trace output into a dedicated file.

Options are given to the driver at startup time, either through the control panels Services applet, on the command line or through the configuration file efdvs.cfg.

## 12 Socket Level Access

### 12.1 Socket Connection

#### Socket Connection and frame definition

For the communication between the application and the driver process with direct socket level access, local sockets are used as communication media.

An application may leave out the runtime library by implementing a direct local socket connection to the driver .The driver is listening on port 8137 by default.

possible access (UNIX style):

```
struct sockaddr_in sa;

/* construct socket */
if((m_socket = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
{
    /* construct socket failed */
    exit;
}

/* connect socket on localhost */
sa.sin_addr.s_addr = inet_addr("127.0.0.1");
sa.sin_port = htons(8137);
sa.sin_family = AF_INET;

/* asynchronous connect request */
if(connect(m_socket, (struct sockaddr *) &sa, sizeof(sa)) != 0)
{
    /* connecting to driver failed */
    exit;
}

/* send frames with send */
/* receive frames with recv */
```

On the local socket, data is organized in frames, which have the following common header of five octets length:

```
flags(1) length(4)
```

The flags field is unused and must be set to zero. Frame contains length bytes XML data in UTF-8 format.

send size:

```
#define HEADER_SIZE 5
frameBuffer[0] =0;
size = frameBuffer.size - HEADER_SIZE;
frameBuffer[1] =0;
frameBuffer[2] =0;
frameBuffer[3] = size / 0x100;
frameBuffer[4] = size % 0x100;
```

receive size:

```
#define HEADER_SIZE 5
receiveSize = 0x100 * frameBuffer[3] +frameBuffer[4] + HEADER_SIZE
;
```



## 12.2 EFT Control Protocol

### EFT Control Protocol

The EFT Control Protocol describes the communication between the application and the driver process with direct socket level access. The protocol defines a set of messages, possible message flows and which attributes may or must be supplied with messages. All messages are presented in XML format.

all messages contain the properties as attributes. a message can also hold the receipt as element.

for the list of messages see [Driver-definition.xml](#) message.

for definition of the attributes, see [Driver-definition.xml](#)

request-response pairs represent operations on the protocol layer. instead of the response, also a [exception Message](#) can be sent by the driver in case of error.

Open	<a href="#">open-request</a> , <a href="#">open-response</a>
Close	<a href="#">close-request</a> , <a href="#">close-response</a>
Transaction	<a href="#">transaction-start</a> , <a href="#">transaction-auth</a>
Commit	<a href="#">transaction-finish</a> , <a href="#">transaction-end</a>
Abort	<a href="#">abort</a>
Balance	<a href="#">balance-request</a> , <a href="#">application-info</a> , <a href="#">balance-response</a>
QueryStatus	<a href="#">query-status</a> , <a href="#">status</a>
DeviceControl	<a href="#">device-control-request</a> , <a href="#">device-control-response</a>

Additionally, [status](#) and [exception-messages](#) may be sent driver side in a broadcast fashion.

All messages contain the id attribute, which is assigned by the client application and is returned with subsequent notification and response messages.

## 12.3 driver-definition.xml

### driver-definition.xml

This file describes the messages and attributes for socket level access. this file is located in the installation directory, subdir "Lib"

the elements "method" describe the different messages and the attributes in it.

the elements "attribute" describe the format and length of the attributes.

the definition of the methods and attributes can be found in this help under "Runtime Library/Driver Object/Methods" and "Runtime Library/Driver Object/Properties"

## 12.4 connectproperties-request Message

### connectproperties-request

typical XML to be sent to the driver after a connection has been established

```
<connectproperties-request
  id=" "
  operation-mode="interactive"
  ecrid="9999"
  cashier="1"
  language="de"
  device-id="1"
  printer-width="40"
  receipt-options="64"
  use-mode="0"
  auto-mode="1"
/>
```

for the list of attributes see [Driver-definition.xml](#) message "connectproperties-request" see [Driver-definition.xml](#)

## 12.5 open-request Message

### open-request Message

Requests an [open](#) operation.

typical XML to be sent to the driver.

```
<open-request
  id=" "
  operation-mode="interactive"
  ecrid="9999"
  cashier="1"
  printer-width="40"
  language="de"
  device-id="1"
  receipt-options="0"
  use-mode="0"
  auto-mode="1"
/>
```

for the list of attributes see [Driver-definition.xml](#) message "open-request"

for the description of the attributes see [Open Method](#): Properties in.

for definition of the attributes, see [Driver-definition.xml](#)

until the response is received [application-info Messages](#) are sent, and a [status Message](#) is sent by the driver.

## 12.6 open-response Message

### open-response Message

Reply from the driver to an open-request message.

```
<open-response
  id=" "
  ecrseq="40"
  ecrid="9999"
  actseq="0"
  terminal-id=" "
  time="20051117 10:46:00"
  receipt-copy-count="1"
>
<receipt>
Test Kunde
Teststrasse 358
TestOrt

SCHICHTANFANG

Act-Id:                               3
Attendant:                             1
ECR-Id:                               9999

17.11.2005                             10:46

</receipt>
</open-response>
```

for the list of attributes see [Driver-definition.xml](#) message "open-response"

for the description of the attributes see [Open Method](#): Properties out.

for definition of the attributes, see [Driver-definition.xml](#)

until the response is received [application-info Messages](#) are sent, and a [status Message](#) is sent by the driver.

## 12.7 close-request Message

### close-request Message

Request a [close](#) operation on the EFT device.  
typical XML to be sent to the driver.

```
<close-request
  id=" "
  ecrid="9999"
  cashier="1"
  device-id="1"
  auto-mode="1"
/>
```

for the list of attributes see [Driver-definition.xml](#) message "close-request"  
for the description of the attributes see [Close Method](#) Properties in.  
for definition of the attributes, see [Driver-definition.xml](#)

until the response is received a [status Message](#) is sent by the driver.

## 12.8 close-response Message

### close-response Message

Reply from the driver to a close-request message.

```
<close-response
  id=" "
  ecrid="9999"
  ecrseq="40"
  receipt-copy-count="1"
>
```

```
<receipt>
Test Kunde
Teststrasse 358
TestOrt
```

SCHICHTENDE

Act-Id:	3
Attendant:	1
ECR-Id:	9999

17.11.2005	10:46
------------	-------

```
</receipt>
</close-response>
```

for the list of attributes see [Driver-definition.xml](#) message "close-response"  
for the description of the attributes see [Close Method](#): Properties out.  
for definition of the attributes, see [Driver-definition.xml](#)

until the response is received a [status Message](#) is sent by the driver.

## 12.9 transaction-start Message

### transaction-start Message

Initiates a [transaction](#) operation on the EFT device.

typical XML to be sent to the driver:

```
<transaction-start
  id=" "
```

```

    ecrid="9999"
    cashier="1"
    amount="300"
    amount-orig="0"
    function="debit"
    options="0"
    language="de"
    currency="CHF"
    ref-number=" "
    trx-ref-num=" "
    auth-code=" "
    entry-mode="A"
    track2=" "
    track3=" "
    device-id="1"
    trx-timeout="60"
    card-remove-indicator="0"
    dcc-mode="0"
    dcc-original-date=" "
  />

```

for the list of attributes see [Driver-definition.xml](#) message "transaction-start"  
 for the description of the attributes see [Transaction Method](#): Properties in.  
 for definition of the attributes, see [Driver-definition.xml](#)

until the response is received, [status Message](#) may be sent by the driver.

## 12.10 transaction-auth Message

### transaction-auth Message

Returns EFT data after successful authorization.

```

<transaction-auth
  id=" "
  amount="300"
  auth-code="CBA00Z"
  ref-number="21325001"
  trx-ref-num="45768725285"
  ecrseq="1"
  time="20051117 13:55:00"
  card-number="F86AEB95B196A57E"
  card-expiration="1205"
  pos-entry-mode="A"
  contract-number="5910012"
  terminal-id="36111302"
  application-name="American Express"
  application-key="VEZ40"
  brand-id="40"
  auth-text="BEWILLIGT"
  booking-period="1234"
  encrypted-pan=" "
  referral-phone-number=" "
  aid="A00000000410"
  actseq="2211"
  receipt-copy-count="2"
  dcc-mode="0"
  currency="CHF"
>
<receipt>
TKC EFT Simulator
(testing only)

```

```

5910012
TRANSAKTIONS-BELEG    TRANSACTION-RECEIPT

```

BUCHUNG  
American Express

```
3758 11XXXX XX115          1205 A
Periode:                    1234
Trm-Id:                     36111302
Attendant:                  1
ECR-Id:                     9999
ECR-Seq:                    1
AID:                        A00000000410
Trx.Seq-Nr:                 21325001
Trx.Ref-Nr:                 45768725285
Auth-Code:                  CBA00Z
BEWILLIGT
```

Total-EFT CHF: 3.00

Unterschrift:

.....  
17.11.2005 13:55

```
</receipt>
</transaction-auth>
```

for the list of attributes see [Driver-definition.xml](#) message "transaction-auth"  
for the description of the attributes see [Transaction Method](#): Properties in.  
for definition of the attributes, see [Driver-definition.xml](#)

until the response is received, [status Message](#) may be sent by the driver.

## 12.11 transaction-finish Message

### transaction-finish Message

Sent by the client to commit or rollback a preceeding transaction operation on the same session.

```
<transaction-finish
  id=" "
  outcome="1 "
/>
```

#### Attributes

outcome	boolean	Specifies whether to commit (1) or rollback(0) the transaction.
---------	---------	---

for the list of attributes see [Driver-definition.xml](#) message "transaction-finish"  
for the description of the attributes see [Commit Method](#): Properties in.  
for definition of the attributes, see [Driver-definition.xml](#)

## 12.12 transaction-end Message

### transaction-end Message

Sent by the driver to confirm the outcome of a transaction after a transaction-finish message.

```
<transaction-end
  id=" "
  outcome="1 "
  amount="0 "
  committed-amount="5005 "
/>
```

#### Attributes

outcome	boolean	Specifies whether the transaction was committed(1) or rolledback(0).
---------	---------	--

for the list of attributes see [Driver-definition.xml](#) message "transaction-end"  
 for the description of the attributes see [Commit Method](#): Properties in.  
 for definition of the attributes, see [Driver-definition.xml](#)

## 12.13 abort Message

### **abort Message**

performs an [abort](#) operation.

typical XML to be sent to the driver.

```
<abort
  id=" "
  ecrid="9999 "
/>
```

for the list of attributes see [Driver-definition.xml](#) message "abort"  
 for the description of the attributes see [Abort Method](#): Properties in.  
 for definition of the attributes, see [Driver-definition.xml](#)

## 12.14 balance-request Message

### **balance-request Message**

Requests an [balance](#) operation.

typical XML to be sent to the driver.

```
<balance-request
  id=" "
  operation-mode="interactive"
  ecrid="9999 "
  device-id="1 "
/>
```

for the list of attributes see [Driver-definition.xml](#) message "balance-request"  
 for the description of the attributes see [Balance Method](#): Properties in.  
 for definition of the attributes, see [Driver-definition.xml](#)

until the response is received [application-info Messages](#) are sent by the driver.

## 12.15 balance-response Message

### **balance-response Message**

Reply from the driver to an balance-request message.

```
<balance-response
  id=" " ecrseq="40 "
  balance-time="20051117 13:42:00 "
  receipt-copy-count="1 "
>
<receipt>
TestKunde
Teststrasse 358
TestOrt
```

TAGESABSCHLUSS

TOTAL

17.11.2005

13:42

---

```
</receipt>
</balance-response>
```

for the list of attributes see [Driver-definition.xml](#) message "balance-request"  
 for the description of the attributes see [Balance Method](#): Properties in.  
 for definition of the attributes, see [Driver-definition.xml](#)

until the response is received [application-info Messages](#) are sent by the driver.

## 12.16 query-status Message

### query-status Message

Requests an status message from the driver.

typical XML to be sent to the driver.

```
<query-status
  id=" "
  ecrId="9999"
  device-id="1"
  auto-mode="1"
/>
```

This message is answered with a [status message](#).

## 12.17 device-control-request Message

### device-control-request Message

Requests an extended terminal [device operation](#), like initialization and submission.

```
<device-control-request
  id=" "
  device-id="1"
  device-control-class="1"
  device-control-command="3"
  device-control-data=" "
/>
```

for the list of attributes see [Driver-definition.xml](#) message "device-control-request"  
 for the description of the attributes see [DeviceControl Method](#): Properties in.  
 for definition of the attributes, see [Driver-definition.xml](#)

## 12.18 device-control-response Message

### device-control-response Message

Contains response data from a device-control operation.

```
<device-control-response
  id=" "
  device-control-result="0"
  device-control-data=" "
  receipt-copy-count="0"
>
</device-control-response>
```

the receipt element may contain a receipt

for the list of attributes see [Driver-definition.xml](#) message "device-control-response"  
 for the description of the attributes see [DeviceControl Method](#): Properties in.  
 for definition of the attributes, see [Driver-definition.xml](#)

## 12.19 exception Message

### Exception Message

This message is sent by the driver to indicate a failure condition. If the client has an outstanding operation running on the session, the operation is aborted before the exception message is sent.

```
<exception
  exception-code="504"
  exception-message="Schicht geschlossen"
  id=" "
/>
```

### Attributes

exception-code	integer		See <a href="#">Error Codes and Messages</a>
exception-message	string	optional	Textual description of the problem
native-status	string	optional	Native error code received from the EFT device
native-message	string	optional	Native error message received from the EFT device

## 12.20 status Message

### status Message

The driver notifies the client of device status changes (device event) using this message.

```
<status
  id=" "
  device-status-code="1"
  device-event-code="0"
  device-application-key=" "
  device-aid=" "
  activation-state="1"
/>
```

for the list of attributes see [Driver-definition.xml](#) message "status"  
 for the description of the attributes see [DeviceEvent Event](#)  
 for definition of the attributes, see [Driver-definition.xml](#)

## 12.21 application-info Message

### application-info Message

This message is sent by the driver to announce application table changes to the client.

message after a open-request

```
<application-info
  id=" "
  application-key="VEZ01"
  application-name="Maestro-CH"
  brand-id="1" aid=" "
  currency="CHF"
  terminal-id="000020991463"
  balance-time=" "
  first-app="0"
  application-type="0"
/>
```

after a balance-request, brands with bookings are reported with additional attributes

```
<application-info
  id=" "
  application-key="VEZ40"
  application-name="American Express"
  brand-id="40"
  aid="A00000000410"
  currency="CHF"
  terminal-id="000000000000"
/>
```



```
    first-app="1"  
    application-type="0"  
    period-number="0"  
    total-count="1"  
    total-amount="300"  
    debit-count="1"  
    debit-amount="300"  
  />
```

**Attributes**

first-app="1" indicates the first application of a new application table

for the list of attributes see [Driver-definition.xml](#) message "application-info"

for the description of the attributes see [Application Object](#) Properties.

for definition of the attributes, see [Driver-definition.xml](#)

# Index

## - A -

Abort Method 25  
 AccountIndex 42  
 Acquirer 41  
 Acquirer Identifier 41  
 ActivationState Property 31  
 Application-Info Message 66  
 ApplicationKey Property 31  
 ApplicationLabel 42  
 Applications Collection 44  
 Applications Collection Application Object 45  
 Applications Property 31  
 Async Property 31  
 AuthCode Property 36  
 AutoMode 5, 38

## - B -

Balance Method 24  
 Brand 42

## - C -

C API 51  
 CardExpiration Property 31  
 CashBack 41  
 Cashier Property 32  
 Close Method 21  
 Close-Request Message 61  
 Close-Response Message 61  
 ComfortPay 9  
 Commit Method 23  
 CommitAmount Method 30  
 CommitPartial Method 24  
 Complete Method 28  
 Configuration and Debugging 52  
 Connect Method 28  
 connect, connectproperties-request 59  
 Coupon 8  
 CustomerLanguage 40

## - D -

DCC rates table 25  
 Device Object 46

Device-Control-Response Message 65  
 DeviceEventCode Property 33  
 Devicelist 25  
 Devices Collection 46  
 DeviceStatusCode Property 33  
 Disconnect Method 28  
 Driver Object 19  
 Driver Shutdown 25  
 DriverAddress Property 33  
 DriverControl Method 25  
 Driver-Control-Request Message 65

## - E -

ECR Information Properties 34  
 ECRID 40  
 ECRSeqCounter 42  
 EFT Control Protocol 59  
 Error Codes and Messages 17  
 Exception Code 43  
 Exception Message 66  
 ExceptionCode ExceptionMessage Properties 34

## - G -

gas station 10  
 General Operations 2  
 Get Device 25  
 Get Device Result 25  
 GiftCard 8

## - I -

IccControl 26  
 IFSF 10

## - L -

Language 40  
 Language Property 34  
 Linux 47

## - M -

Mobile Coupon 8  
 Mobile Voucher 7

**- O -**

OnCommandComplete Event 43  
 OnDeviceEvent Event 43  
 OnError Event 43  
 OnStatusChange Event 43  
 Open Close Balance and Submit Operations 5  
 Open Method 20  
 Open-Request 60  
 Open-Response Message 60  
 OperationMode Property 35  
 OSX 48

**- P -**

PAN key entry 23  
 PARActor 40  
 PARActorPwd 40  
 PARAmount 40  
 PARCPMethod 40  
 PARCurrC 40  
 PARCurrE 40  
 PARRefNum 40  
 partial approval 23  
 partial Commit 23, 30  
 Petrol 10, 11, 12, 13, 15  
 Petrol Eject Card 13  
 Petrol Key Code 13  
 Petrol PIN Check 12  
 Petrol Sale Item 15  
 Petrol Transaction Flow 13  
 Print 40  
 PrinterWidth 35  
 PrintOnEFT Method 29, 30, 39

**- Q -**

QueryStatus Method 25

**- R -**

Receipt 35, 40  
 Receipt Object 45  
 Receipt Properties 35  
 ReceiptCopyCount 35  
 ReceiptMerchantText 35  
 ReceiptOptions 35  
 ReceiptText 35  
 RefNumber Property 36

Release Device 25  
 Runtime Library 17

**- S -**

Static Key PAN Receipt Index 41  
 StatKeyPANRctInd 41  
 Status Message 66  
 Status Property 37  
 SubBrand 42  
 Submission 25

**- T -**

TerminalID 40  
 Time Property 37  
 Timeouts 18  
 Tip 39  
 TipAmount 39  
 TopUp 7, 25, 37  
 TopUp Voucher Services 25  
 Trace and Debug Options 57  
 Track2 Property 38  
 Transaction Method 21  
 Transaction Operation 2  
 Transaction Options 23  
 Transaction-Auth Message 62  
 Transaction-End Message 63  
 Transaction-Finish Message 63  
 Transaction-Start Message 61  
 TrxRefNumber Property 36

**- U -**

Using the COM/OLE Interface 48  
 Using the Java interface 48

**- V -**

Validation Method 24  
 Voucher Services 25

**- W -**

Welcome 1  
 Windows 47

