Payment Services

TIMApi Implementation Guide

1.2

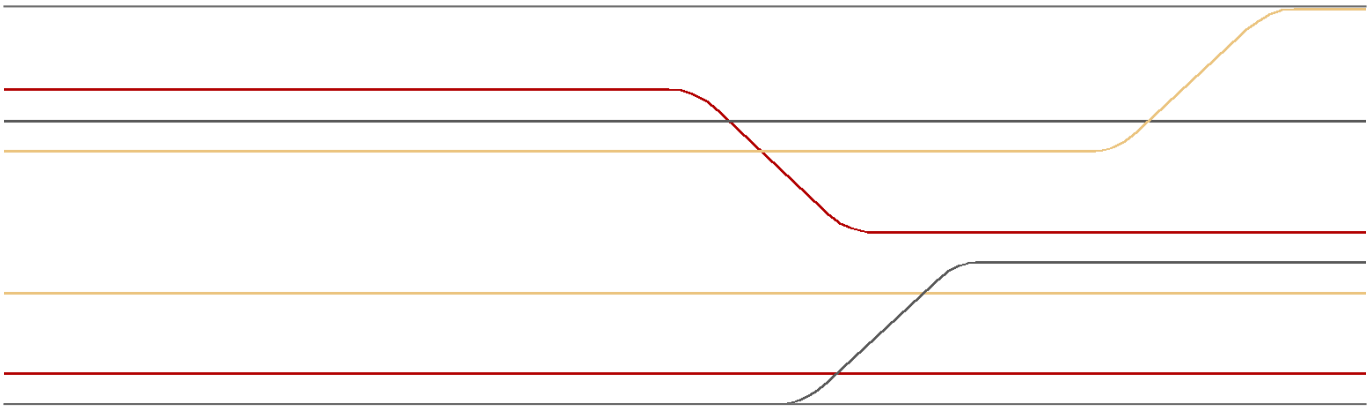


Table of Contents

Generated: February 07, 2017

Version Table

Version	1.2
Distribution	08.12.2016
Status	Final
Author	Roland Plüss
Classification	internal

Approved By

Function	Name	Date

Amendment Control

Version	Author	Date	Content
1.0	B. Vontobel, Hp. Lutz, R. Plüss, M. Beutler, C. Anderegg	09.11.2016	Release version
1.1	R. Plüss	22.11.2016	Updated Post-Automatisation Parameters in TerminalSettings and updated Configuration File available Parameters Added InitTransactionAsync method call. Added InitTransactionCompleted TerminalNotifier method.
1.2	R. Plüss	08.12.2016	Added TransactionType to TransactionResponse.

Table 1: Amendment control

Open Points

No	Description	PIC	Date / Priority
1	Adjust .NET & Swift implementation examples to be up to date. Add asynchronous examples.		
2	Examples for compact, supercompact, ultracompact receipts in appendix.		
3	Split documentation. E.g. “Basic-Retail”		
4	Add sequence diagrams.		

Table 2: Open Points

Constraints

No	Description	PIC	Date / Priority
1			

Table 3: Constraints

Dependencies

Dependency	Description

Table 4: Dependencies

1 Overview

1.1 Introduction

The TIM API is the standard application interface for ECR integration using SIX Payment Services terminals.

TIM API provides a comprehensive feature set to support the requirements of several different markets. This guideline is intended to be used by ECR integrators that need to implement the SIXeftAPI functionality in their products.

The following document contains an overview structure, description of the TIM API modes synchronous/asynchronous and describes the corresponding functions, notifiers and data elements that can be used with the TIM API. Furthermore implementation examples are provided for Java, .NET and Swift language. Nevertheless always the corresponding descriptions of a function or data element are authoritative, not the examples.

For further information about the architecture of the TIM API can be found in [B9], this document is meant to cover only implementation purposes.

1.2 Reference

1.2.1 Basis Documents

Ref.	Document	Version
[B1]	EP2 Specification	6.3.0
[B4]	ISO 639-1 Language codes	2002
[B5]	ISO 3166-1 alpha-2 Country codes	2013
[B6]	RFC 5646 Language codes	2009
[B7]	ISO 4217 Currency codes	2015
[B8]	ISO 10646 UTF-8	2012
[B9]	TIM & SIXml Architecture: The SIX ECR Integration System	0.10

Table 5: Basis Documents

2 System Overview

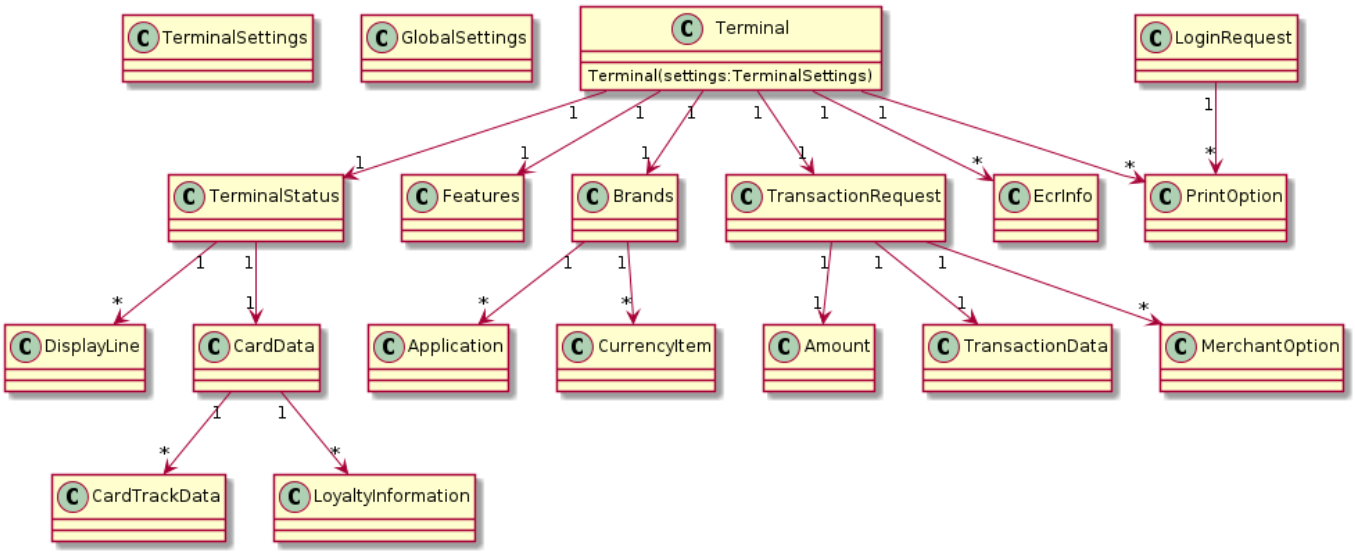
The following chapters describe a brief system overview.

2.1 Class Overview

The following diagrams show a simplified class overview without having the demand of completeness. All corresponding class members are mentioned in the specified chapter of each class. This chapter is only meant to show the relations between the classes used in the TIM API.

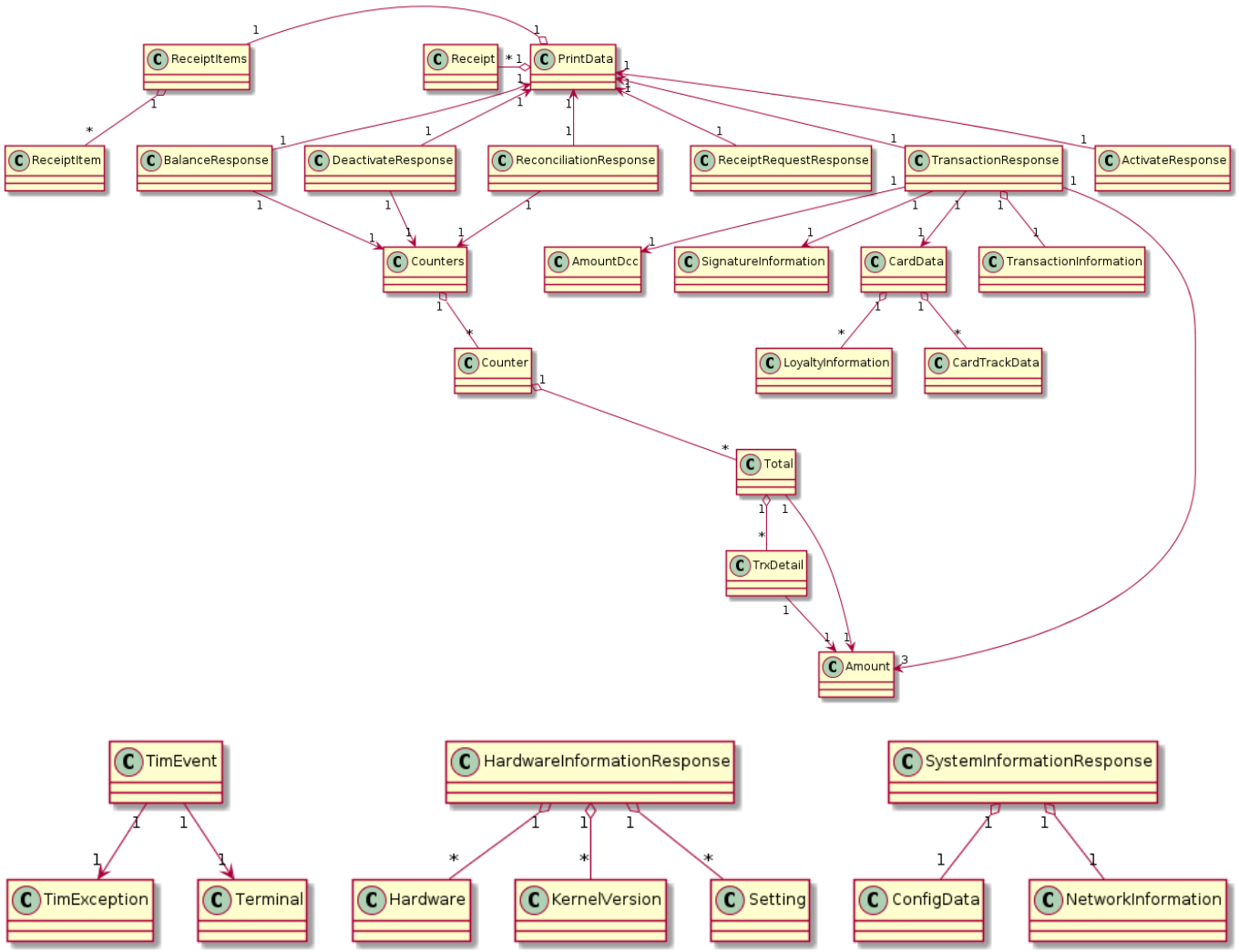
2.1.1 Terminal class

This diagram shows all classes that are connected and therefore used with the *Terminal* class.



2.1.2 Response classes returned from functions

This diagram shows the relations of the classes that are used as return value of the TIM API functions.



2.2 Setup Configuration

The TIMApi module can communicate with the Terminal over different communication channels. The default communication is over TCP/IP connection.

For TCP/IP connection there are four possible ways to connect to the terminal:

1. The TIMApi module knows the Terminal ID (TID) of the terminal. In this case the TIMApi module finds the terminal with a broadcast mechanism.
    1. The TIMApi module starts broadcasting and connects to the connection information send by the matching terminal.
    2. The TIMApi module starts broadcasting with connection information waiting for the terminal to connect to the TIMApi module
2. The TIMApi module knows the IP address of the terminal.
    1. The TIMApi module connects to the terminal using the known IP address
    2. The TIMApi waits for the terminal to connect using the known IP address

The configuration can be done in two ways:

1. In a configuration file “TimApi.cfg”
2. Using the API TerminalSettings done from the application.

For further information's refer to TerminalSettings.

2.3 Operations Overview

1. Create TerminalSettings and initialize connection parameters
2. Create Terminal instance using the created TerminalSettings
3. Set some properties if needed:
    - PosId
    - List<EcrInfo>
    - List<PrintOption>
4. Connect and Login. This can be done automatically, when the first Terminal operation is called or manually using the Login Method
5. Use Terminal Operation Methods. A basic flow to perform a transaction is:

2.4 Method Overview

The methods are in the **Terminal** Class

- Login
  the Login operation connects to the terminal and exchanges some properties.

- Activate and Deactivate
  The Activate and Deactivate operations are used to change the EFT shift state. They are carried out when the operator is logged on, resp. off the ECR. The Activate operation must be performed before transactions can be carried out.

- Transaction, InitTransaction, Cancel, Commit, Rollback
  The Transaction operation is used for all types of financial operation carried out on the device. A set of types of this operation exists. A running transaction can be aborted with the Cancel operation. A Transaction is finished with the Commit operation or rolled back with the Rollback operation.

- Balance
  The Balance operation requests per-contract counter totals from the device and triggers an end-of-day procedure.

Administrative Operations

- ApplicationInformation
- ChangeSettings
- CounterRequest
- DccRates
- HardwareInformation
- Login
- Logout
- Reboot
- Reconciliation
- ReceiptRequest
- Reconfig
- SoftwareUpdate
- SystemInformation

2.5 Synchronous, asynchronous behavior and events

The basic operation mode of the **Terminal** Methods are:

1. Synchron
   method returns after the operation is finished successful and throws a TimException otherwise.

2. Asynchron
   method returns immediately after the operation has started successfully or throws a TimException otherwise. All methods receive a TimEvent. In case of failure a TimException is included.
   - ActivateCompleted - operation started by ActivateAsync finished.
   - ApplicationInformationCompleted - operation started by ApplicationInformationAsync finished.
   - BalanceCompleted - operation started by BalanceAsync finished.
   - ChangeSettingsCompleted - operation started by ChangeSettingsAsync finished.
   - CommitCompleted - operation started by CommitAsync finished.
   - CounterRequestCompleted - operation started by CounterRequestAsync finished.
   - DeactivateCompleted - operation started by DeactivateAsync finished.
   - DccRatesCompleted - operation started by DccRatesAsync finished.
   - HardwareInformationCompleted - operation started by HardwareInformationAsync finished.
   - InitTransactionCompleted - operation started by InitTransactionAsync finished.
   - LoginCompleted - operation started by LoginAsync finished.
   - LogoutCompleted - operation started by LogoutAsync finished.
   - RebootCompleted - operation started by RebootAsync finished.
   - ReconciliationCompleted - operation started by ReconciliationAsync finished.
   - ReceiptRequestCompleted - operation started by ReceiptRequestAsync finished.
   - ReconfigCompleted - operation started by ReconfigAsync finished.
   - RollbackCompleted - operation started by RollbackAsync finished.
   - SoftwareUpdateCompleted - operation started by SoftwareUpdateAsync finished.
   - SystemInformationCompleted - operation started by SystemInformationAsync finished.
   - TransactionCompleted - operation started by TransactionAsync finished.
   - RequestCompleted - operation started by an asynchronous method finished. Applications can use both operation specific completion method in combination with this generic completion method.
   - TerminalStatusChanged - Terminal status has changed. The new status can be retrieved from the TerminalStatus property.

Methods called on Terminal perform synchronous by default. The asynchronous method has the same name with *Async* appended to its name.

---

3 TIMApi Application Interface

The following sections describe the TIMApi application interface. The interface description is programming language and platform independent. The actual implementation for each supported language and platform does vary. The following terms are used in this document.

- **Class**
  A data structure containing properties and method. In general classes are not thread-safe.

- **Immutable Class**
  A data structure containing read-only properties and methods. Once created the properties of the immutable class instance can not be changed any more. Immutable classes are thread-safe.

- **Enumeration**
  A data type of integer or string type with restricted set of possible values. The value of an enumeration entry can be a single value or a complex type with multiple values if useful. Enumerations are immutable and thread-safe. Depending on the language used an enumeration is implemented as a real enumeration or a mapping between possible values and the immutable enumeration constants.

- **Property**
  A member of a class storing an object parameter of a specific type. Properties can be changed. In general they are not thread-safe. Depending on the language used a property is implemented as a member the user can directly access (C# for example *object.propertyX = 8*) or using getters and setters (Java for example *object.getPropertyX()* and *object.setPropertyX(8)*).

- **Immutable Property**
  A member of an immutable class storing an object parameter of a specific type. Immutable properties can not be changed. They are thread-safe.

- **Method**
  A callable method on a class or immutable class. In general methods on classes are not thread-safe. Methods on immutable classes are thread-safe.

- **Notifier**
  A notification receiver implemented by the user. Depending on the language used a notifier is implemented as event (C# for example) or a listener (Java for example). In general notifiers are invoked from a thread other than the main thread. Only thread-safe properties and methods are allowed to be used inside notifiers. If not thread-safe functionality is required the user has to use language specific methods to react to the notification in the main thread.

- **Exception**
  An exception thrown in called methods. Aborts the method handing control to the language specific exception handling mechanism.

3.1 Global Settings

The global configuration can be done on two ways:

1. In a configuration file “TimApi.cfg”
2. With the API “GlobalSettings” done from the application

The singleton class **GlobalSettings** allows the setting from Application. It can be done before the first *Terminal* instance is created or afterwards.

All configuration has to be done before the first method call in a newly created Terminal Object.

GlobalSettings class properties:

- String **LogDir**
  Name of logging directory

3.2 Terminal Settings

The per terminal configuration can be done on two ways:

1. In a configuration file "TimApi.cfg"
2. With the API *TerminalSettings* done from the application

The class *TerminalSettings* allows the setting from Application. Each instance of the *Terminal* class is initialized using an own instance of *TerminalSettings* containing the configuration relevant for this terminal.

All configuration has to be done before an instance of *TerminalSettings* is used to create a *Terminal* class instance.

Defines also automatic behaviour for interacting with EFT terminals. By default all automatic behaviours are enabled.

The *TerminalSettings* class has two constructors:

- TerminalSettings(): An empty constructor that uses the global ini-section of the TimApi.cfg file, if present.
- TerminalSettings(String:DeviceId): Constructor with a user defined *DeviceId* string parameter which uses the configuration in the TimApi.cfg file with the same *DeviceId* if present. Otherwise it uses the global ini-section.

TerminalSettings class properties:

- String **TerminalId**
  Terminal ID to be broadcasted in case of ConnectionMode **Broadcast**

- ConnectionMode **ConnectionMode**
  Broadcast (**default**) or OnFixIP

- String **ConnectionIPString**
  IP address of the EFT terminal in case of ConnectionMode **OnFixIP**.

- int **ConnectionIPPort**
  Listening Port of the EFT terminal in case of ConnectionMode **OnFixIP**

- ProtocolType **ProtocolType**
  Protocol type

- String **Logdir**
  Defines where the Log File has to be generated. Overrides the LogDir defined in GlobalSettings

- boolean **AutoConnect**
  Automatically connects after a new instance of Terminal API is created or after the connection to the EFT has been lost. Does not automatically connect if the API user calls Disconnect() manually. Default is on.

- boolean **AutoLogin**
  Automatically logs in after connected to the EFT. Default is on.

- boolean **FetchBrands**
  Automatically retrieves application information during logging in. Default is on.

- boolean **AutoActivate**
  Automatically activate EFT after every request finishes that is applicable in logged in state. This affects any situation such a request is issued. Also automatically activating after logging in. Does not affect calling Deactivate() manually. Default is on.

- boolean **AutoCommit**
  After executing the Transaction- function the API commits the transaction automatically. Default is on.

- boolean **AutoDeactivate**
  Automatically deactivate EFT after every request finishes that is applicable in open state. This affects any situation such a request is issued. Does not affect calling Deactivate() manually. Default is on.

- boolean **AutoLogout**
  Automatically logs out after Deactivate() has been called or an Auto-Deactivate has been performed. Default is on.

- boolean **AutoDisconnect**
  Automatically disconnect after Logout() has been called or an Auto-Logout has been performed. Default is on.

—-

3.3 Terminal Members

In *Terminal* instances a set of properties are accessible.

Properties showing the Status of the Terminal

- TerminalStatus **TerminalStatus**

Properties to be set before first Login

- List<EcrInfo> **EcrData**
- List<PrintOption> **PrintOptions**
- String **PosId**
  Cash Register identifier
- String **UserId**
  User identifier
- int **ManufacturerFlags**

Immutable properties available after the first login

- **immutable** String **TerminalId**
  EFT Terminal identifier.
- **immutable** Features **Features**
- **immutable** int **ProtocolLevel**
  Protocol level used by the EFT Terminal. Depends on the used protocol.
- **immutable** Brands **Brands**
- **immutable** ConfigData **ConfigData**
  EFT Terminal configuration data.

Immutable properties available after first Activate

- **immutable** int **ActSeqCounter**
  Activation sequence counter, Incremented for every new Activation performed.

Properties to be set for Transaction

- TransactionRequest **TransactionRequest**

Properties used to format receipts from receipt fields.

- ReceiptFormatter **ReceiptFormatter**

3.3.1 ReceiptFormatter

The *ReceiptFormatter* interface is responsible to create final receipts from receipt fields. It is used if the ECR requests to format receipts itself. The interface has only one method.

List<Receipt> **FormatReceipt(** RequestType **requestType**, ReceiptItems **receiptItems)**

parameters	RequestType	Type of request requesting the receipt to be created.
	ReceiptItems	Receipt items used to create receipt.
returns	List<Receipt>	List of receipts generated from the receipt fields.

Table 6: Method FormatReceipt

The following receipt formatter implementations are available.

- **NormalReceiptFormatter (default)**. Formats receipts using normal receipt format.
- **CompactReceiptFormatter**. Formats receipts using compact receipt format.
- **SuperCompactReceiptFormatter**. Formats receipts using super compact receipt format.
- **UltraCompactReceiptFormatter**. Formats receipts using ultra compact receipt format.

The TerminalSettings define the receipt formatter to use.

3.4 Terminal Class Methods

The following chapters describe the methods of the *Terminal* class, which are available for the retail functionality. They are grouped into 3 types:

- Common functions - Equal function for synchronous and asynchronous mode
- Synchronous functions
- Asynchronous functions

For each function examples are provided. Nevertheless always the tables which describe the functions and their parameters in the corresponding sections are authoritative. Examples are for illustration only.

3.4.1 Common

The following functions are independent from the API mode sync or async.

3.4.1.1 Constructor

Create new instance of the Terminal class hand over a prepared TerminalSettings instance. Once created the terminal settings can not be changed any more for this terminal instance.

Terminal(TerminalSettings **settings)**

parameters	TerminalSettings	Terminal settings loaded from configuration final and modified by API calls
throws	TimException	

Table 7: Method Terminal Constructor

3.4.1.2 Cancel

The *Cancel*-function is used to abort an open asynchronous Financial Transaction or Non-Financial Transaction request, except for a Commit or RollbackAsync request, which cannot be cancelled.

void **Cancel()**

parameters	none
return	none
throws	TimException

Table 8: Method Cancel

Further information:
A *Cancel*-request is a best effort request. The EFT Terminal can ignore the *Cancel*-request if the request to be cancelled is in a state it can not be cancelled. A *Cancel*-request has no effect if there's no open request of a Financial Transaction or Nonfinancial Transaction. If the EFT Terminal cancels the request in progress an error response is send back for the cancelled request (see chapter Abort Purchase for an example). *Cancel*-requests can be send multiple times in an attempt to cancel a request in progress but should be spaced by a few seconds. *Cancel*-requests themselves are never acknowledged by the EFT Terminal.

The Transaction request can only be cancelled before a Commit has been performed. Also the underlying payment protocol can restrict this functionality.

3.4.1.3 Connect

Initiates a connection to the EFT Terminal. Calling the Connect-method is not needed normally, because all methods make first a connect, if not connected.

void **Connect()**

parameters	none
return	none
throws	TimException

Table 9: Method Connect

3.4.1.4 Disconnect

The *Disconnect*-function interrupts the connection to the EFT Terminal.

void **Disconnect()**

parameters	none
return	none
throws	TimException

Table 10: Method Disconnect

3.4.2 Synchronous functions

The following functions are used for a synchronous processing.

3.4.2.1 Activate

The *Activate*-function opens a user shift. If the shift is already open, no error is returned.

ActivateResponse **Activate()**

parameters	none	
return	ActivateResponse	contains activation sequence counter and print information for the merchant
throws	TimException	

Table 11: Method Activate

3.4.2.2 ApplicationInformation

The *ApplicationInformation*-function requests the list of brands available on the terminal.

Side-Effect: Updates Brands member with all brands available on the terminal. Use Terminal.getBrands() to retrieve them.

void **ApplicationInformation()**

parameters	none
return	none

throws	TimException	

Table 12: Method ApplicationInformation

3.4.2.3 Balance

The *Balance*-function is used to force the EFT Terminal to transmit all transactions to the host system as well to do the daily closing.

BalanceResponse **Balance()**

parameters	none	
return	BalanceResponse	contains counters and print information for the merchant
throws	TimException	

Table 13: Method Balance

3.4.2.4 ChangeSettings

The *ChangeSettings*-function changes configuration parameters of the EFT Terminal.

void **ChangeSettings(** List<Setting> **settings)**

parameters	settings	List of settings to change
return	none	
throws	TimException	

Table 14: Method ChangeSettings

3.4.2.5 Commit

Performs *Commit*-operation after a successful Transaction call.

void **Commit()**

parameters	none	
return	none	
throws	TimException	

Table 15: Method Commit

3.4.2.6 CounterRequest

The *CounterRequest*-function is used to get counter information's from the EFT Terminal.

Counters **CounterRequest(** CounterType **type)**

parameters	type	type of counters to request
return	Counters	contains counters
throws	TimException	

Table 16: Method CounterRequest

3.4.2.7 Deactivate

The *Deactivate*-function closes a user shift. If the shift is already closed, no error is returned.

DeactivateResponse **Deactivate()**

parameters	none	
return	DeactivateResponse	contains counters and print information for the merchant
throws	TimException	

Table 17: Method Deactivate

3.4.2.8 DccRates

The *DccRates*-function requests DCC rates from the EFT Terminal. The DCC rates are returned as receipt.

PrintData **DccRates()**

parameters	none	
return	PrintData	contains print information for merchant
throws	TimException	

Table 18: Method DccRates

3.4.2.9 HardwareInformation

The *HardwareInformation*-function is used to get hardware information from the EFT Terminal.

void **HardwareInformation()**

parameters	none	
return	HardwareInformationResponse	contains information about EFT Terminal hardware and EMV kernels and configuration parameters supported by the EFT Terminal
throws	TimException	

Table 19: Method HadwareInformation

3.4.2.10 InitTransaction

The *InitTransaction*-function is used to initialize a transaction knowing the amount or the card type.

void **InitTransaction()**

parameters	none	
return	none	
throws	TimException	

Table 20: Method InitTransaction

3.4.2.11 Login

The *Login*-function is used to activate a communication session between the ECR and the terminal.

Note: Before calling set the print options, POS identifier and manufacturer flags set in the terminal instance.

Side-Effect: After completing the request updates the features, brands and terminal identifier in the terminal instance. Fetching these information can be disabled if *Auto-FetchBrands* is disabled in TerminalSettings.

void **Login()**

parameters	none
return	none
throws	TimException

Table 21: Method Login

void **Login(** LoginRequest **request)**

parameters	request	contains protocol options, print options, POS identifier and custom manufacturer flags instead of taking them from the terminal properties.
return	none	
throws	TimException	

Table 22: Method Login

3.4.2.12 Logout

The *Logout*-function terminates an active communication session between the ECR and the terminal.

void **Logout()**

parameters	none
return	none
throws	TimException

Table 23: Method Logout

3.4.2.13 Reboot

The *Reboot*-function is used to force the EFT Terminal to reboot.

void **Reboot()**

parameters	none
return	none
throws	TimException

Table 24: Method Reboot

3.4.2.14 Reconciliation

The *Reconciliation*-function is used to force the EFT Terminal to transmit all financial transactions to the host system.

ReconciliationResponse **Reconciliation()**

parameters	none	
return	ReconciliationResponse	contains counters and print information for the merchant
throws	TimException	

Table 25: Method Reconciliation

3.4.2.15 ReceiptRequest

The *ReceiptRequest*-function is used to receive the latest receipt or a list of silent receipts.

ReceiptRequestResponse **ReceiptRequest(** ReceiptRequestType *ReceiptType* **)**

parameters	ReceiptRequestType	type of receipts to request from the EFT Terminal
return	ReceiptRequestResponse	contains all requested receipts up to a maximum number of receipts the EFT Terminal can send. if HasMoreReceipts is *True* not all receipts could be send by the EFT Terminal. call *ReceiptRequest* again to obtain more receipts
throws	TimException	

Table 26: Method ReceiptRequest

3.4.2.16 Reconfig

The *Reconfig*-function is used to force the EFT Terminal to get the configuration from the service center.

PrintData **Reconfig()**

parameters	none	
return	PrintData	contains print information for the merchant
throws	TimException	

Table 27: Method Reconfig

3.4.2.17 Rollback

The *Rollback*-function prevents a transaction from being committed to the transaction log and generates a technical reversal of the authorization. For payment protocols supporting the function a declined receipt may be generated.

PrintData **Rollback()**

parameters	none	
return	PrintData	contains print information for the merchant and cardholder
throws	TimException	

Table 28: Method Rollback

3.4.2.18 SoftwareUpdate

The *SoftwareUpdate*-function is used to force the EFT Terminal to start a Software Update.

int **SoftwareUpdate()**

parameters	none	
return	UpdateStatus	contains update status
throws	TimException	

Table 29: Method SoftwareUpdate

3.4.2.19 SystemInformation

The *SystemInformation*-function is used to request system information from the EFT Terminal.

Note: Set EcrData property before calling *SystemInformation*.

SystemInformationResponse **SystemInformation()**

parameters	none	
return	SystemInformationResponse	Contains configuration and network information
throws	TimException	

Table 30: Method SystemInformation

3.4.2.20 Transaction

The *Transaction*-function starts an EFT Terminal Transaction.

Note: The transaction parameters are taken from TransactionRequest terminal property since those do not change often if at all (default parameters).

TransactionResponse **Transaction(** TransactionType **type**, Amount **amount)**

parameters	type	a Financial Transaction Function
	amount	transaction amount
return	TransactionResponse	contains transaction result information and print information for the merchant and cardholder
throws	TimException	

Table 31: Method Transaction

TransactionResponse **Transaction(** TransactionType **type**, TransactionRequest **request)**

parameters	type	a Financial Transaction Function
	request	transaction parameters without using TransactionRequest property
return	TransactionResponse	contanis transaction result information and print information for the merchant and cardholder
throws	TimException	

Table 32: Method Transaction

3.4.3 Asynchronous functions

The following functions shall be used for asynchronous API mode.

3.4.3.1 ActivateAsync

The *ActivateAsync*-function opens a user shift. If the shift is already open, no error is returned.
Asynchronous version of *Activate*-function. Returns immediately and calls ActivateCompleted on all notifiers once finished.

void **ActivateAsync()**

parameters	none
return	none
throws	TimException

Table 33: Method ActivateAsync

3.4.3.2 ApplicationInformationAsync

The *ApplicationInformationAsync*-function requests the list of brands available on the terminal.
Asynchronous version of *ApplicationInformation*-function. Returns immediately and calls ApplicationInformationCompleted on all notifiers once finished.

Side-Effect: Updates Brands member with all brands available on the terminal. Use Terminal.getBrands() to retrieve them.

void **ApplicationInformationAsync()**

parameters	none
return	none
throws	TimException

Table 34: Method ApplicationInformationAsync

3.4.3.3 BalanceAsync

The *BalanceAsync*-function is used to force the EFT Terminal to transmit all transactions to the host system as well to do the daily closing.

Asynchronous version of *Balance*-function. Returns immediately and calls BalanceCompleted on all notifiers once finished.

void **BalanceAsync()**

parameters	none
return	none
throws	TimException

Table 35: Method BalanceAsync

3.4.3.4 ChangeSettingsAsync

The *ChangeSettingsAsync*-proces configuration parameters of the EFT Terminal.

Asynchronous version of *ChangeSettings*-function. Returns immediately and calls ChangeSettingsCompleted on all notifiers once finished.

void **ChangeSettingsAsync(** List<Settings> **settings)**

parameters	settings	List of settings to change
return	none	
throws	TimException	

Table 36: Method ChangeSettingsAsync

3.4.3.5 CommitAsync

Performs *Commit*-operation after a successful call to Transaction or TransactionAsync method.

Asynchronous version of *Commit*-function. Returns immediately and calls CommitCompleted on all notifiers once finished.

void **CommitAsync()**

parameters	none
return	none
throws	TimException

Table 37: Method CommitAsync

3.4.3.6 CounterRequestAsync

The *CounterRequestAsync*-function is used to get counter information's from the EFT Terminal.

Asynchronous version of *CounterRequest*-function. Returns immediately and calls CounterRequestCompleted on all notifiers once finished.

void **CounterRequestAsync(** CounterType **type)**

parameters	type	type of counters to request
return	none	
throws	TimException	

Table 38: Method CounterRequestAsync

3.4.3.7 DeactivateAsync

The *DeactivateAsync*-function closes a user shift. If the shift is already closed, no error is returned.

Asynchronous version of *Deactivate*-function. Returns immediately and calls DeactivateCompleted on all notifiers once finished.

void **DeactivateAsync()**

parameters	none
return	none
throws	TimException

Table 39: Method DeactivateAsync

3.4.3.8 DccRatesAsync

The *DccRatesAsync*-function requests DCC rates from the EFT Terminal. The DCC rates are returned as receipt.

Asynchronous version of *DccRates*-function. Returns immediately and calls DccRatesCompleted on all notifiers once finished.

void **DccRatesAsync()**

parameters	none
return	none
throws	TimException

Table 40: Method DccRatesAsync

3.4.3.9 HardwareInformationAsync

The *HardwareInformationAsync*-function is used to get hardware information from the EFT Terminal.

Asynchronous version of *HardwareInformation*-function. Returns immediately and calls HardwareInformationCompleted on all notifiers once finished.

void **HardwareInformationAsync()**

parameters	none
return	none
throws	TimException

Table 41: Method HardwareInformationAsync

3.4.3.10 InitTransactionAsync

The *InitTransaction-function* is used to initialize a transaction knowing the amount or the card type.

Asynchronous version of *InitTransaction*-function. Returns immediately and calls InitTransactionCompleted on all notifiers once finished.

void **InitTransactionAsync()**

parameters	none
return	none
throws	TimException

Table 42: Method InitTransactionAsync

3.4.3.11 LoginAsync

The *LoginAsync*-function is used to activate a communication session between the ECR and the terminal.

Note: Before calling set the print options, POS identifier and manufacturer flags properties in the terminal instance.

Side-Effect: After completing the request updates the features, brands and terminal identifier is set in the terminal instance. Fetching these information can be disabled with *Auto-FetchBrands* in TerminalSettings.

Asynchronous version of *Login*-function. Returns immediately and calls LoginCompleted on all notifiers once finished.

void **LoginAsync()**

parameters	none
return	none
throws	TimException

Table 43: Method LoginAsync

void **LoginAsync(** LoginRequest **request)**

parameters	request	contains protocol options, print options, POS identifier and custom manufacturer flags instead of taking them from the terminal properties.
return	none	
throws	TimException	

Table 44: Method LoginAsync

3.4.3.12 LogoutAsync

The *LogoutAsync*-function terminates an active communication session between the ECR and the terminal.

Asynchronous version of *Logout*-function. Returns immediately and calls LogoutCompleted on all notifiers once finished.

void **LogoutAsync()**

parameters	none
return	none
throws	TimException

Table 45: Method LogoutAsync

3.4.3.13 RebootAsync

The *RebootAsync*-function is used to force the EFT Terminal to reboot.

Asynchronous version of *Reboot*-function. Returns immediately and calls RebootCompleted on all notifiers once finished.

void **RebootAsync()**

parameters	none
return	none
throws	TimException

Table 46: Method RebootAsync

3.4.3.14 ReconciliationAsync

The *ReconciliationAsync*-function is used to force the EFT Terminal to transmit all financial transactions to the host system.

Asynchronous version of *Reconciliation*-function. Returns immediately and calls ReconciliationCompleted on all notifiers once finished.

void **ReconciliationAsync()**

parameters	none
return	none
throws	TimException

Table 47: Method ReconciliationAsync

3.4.3.15 ReceiptRequestAsync

The *ReceiptRequestAsync*-function is used to receive the latest receipt or a list of silent receipts.

void **ReceiptRequestAsync(** ReceiptRequestType *ReceiptType* **)**

Asynchronous version of *ReceiptRequest*-function. Returns immediately and calls ReceiptRequestCompleted on all notifiers once finished.

parameters	ReceiptRequestType	type of receipts to request from the EFT Terminal
return	none	
throws	TimException	

Table 48: Method ReceiptRequestAsync

3.4.3.16 ReconfigAsync

The *ReconfigAsync*-function is used to force the EFT Terminal to get the configuration from the service center.

Asynchronous version of *Reconfig*-function. Returns immediately and calls ReconfigCompleted on all notifiers once finished.

void **ReconfigAsync()**

parameters	none
return	none
throws	TimException

Table 49: Method ReconfigAsync

3.4.3.17 RollbackAsync

The *RollbackAsync*-function prevents a transaction from being committed to the transaction log and generates a technical reversal of the authorization. For payment protocols supporting the function a declined receipt may be generated.

Asynchronous version of *Rollback*-function. Returns immediately and calls RollbackCompleted on all notifiers once finished.

void **RollbackAsync()**

parameters	none
return	none
throws	TimException

Table 50: Method RollbackAsync

3.4.3.18 SoftwareUpdateAsync

The *SoftwareUpdateAsync*-function is used to force the EFT Terminal to start a Software Update.

void **SoftwareUpdateAsync()**

parameters	none
return	none
throws	TimException

Table 51: Method SoftwareUpdateAsync

3.4.3.19 SystemInformationAsync

The *SystemInformationAsync*-function is used to request system information from the EFT Terminal.

Asynchronous version of *SystemInformation*-function. Returns immediately and calls SystemInformationCompleted on all notifiers once finished.

Note: Set EcrData property before calling *SystemInformation*.

void **SystemInformationAsync()**

parameters	none
return	none
throws	TimException

Table 52: Method SystemInformationAsync

3.4.3.20 TransactionAsync

The *TransactionAsync*-function starts an EFT Terminal Transaction.

Asynchronous version of *Transaction*-function. Returns immediately and calls TransactionCompleted on all notifiers once finished.

Note: The transaction parameters are taken from TransactionRequest terminal property since those do not change often if at all (default parameters).

void **TransactionAsync(** TransactionType **type**, Amount **amount)**

parameters	type	a Financial Transaction Function
	amount	transaction amount
return	none	
throws	TimException	

Table 53: Method Transaction

void **TransactionAsync(** TransactionType **type**, TransactionRequest **request)**

parameters	type	a Financial Transaction Function
	request	transaction parameters without using TransactionRequest property
return	none	
throws	TimException	

Table 54: Method TransactionAsync

3.5 Terminal notifier

The following chapters describe the methods of the notifier interface used for asynchronous requests. Each of the *Async methods invoke the matching notifier method upon completing successfully or with failure. The invoked notifier method can be implemented to react to the specific request type. If not implemented the default implementation invokes the generic RequestCompleted notifier method. Implementing this method allows to react to any kind of completed request.

The first parameter of each notifier method is an instance of TimEvent. It contains information about the terminal instance sending the event, the event type and error information if the request failed. Some notifier methods feature an additional data parameter containing the result of the asynchronous request. This parameter matches the return value of the synchronous method counterpart of the request. The generic RequestCompleted has a generic data parameter required to be converted to the appropriate data type for the request in question. It can be None-Value if the request in question does not have a result value.

Multiple notifiers can be registered to provide distinct behaviour. Any number of methods in the notifier can be implemented. Both the request specific methods and the generic RequestCompleted method can be implemented in the same notifier for complex use cases.

The order in which the notifiers are invoked is undefined. User are discouraged to rely on the order notifiers are invoked.

3.5.1 ActivateCompleted

The *ActivateCompleted* method is called if an ActivateAsync request finished. If not implemented invokes RequestCompleted.

void **ActivateCompleted(** TimEvent **event**, ActivateResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains activation sequence counter and print information for the merchant.

Table 55: Method ActivateCompleted

3.5.2 ApplicationInformationCompleted

The *ApplicationInformationCompleted* method is called if an ApplicationInformationAsync request finished. If not implemented invokes RequestCompleted.

Note: The brands member in the terminal instance is updated before the first notifier is called. Use event.getTerminal().getBrands() to retrieve them.

void **ApplicationInformationCompleted(** TimEvent **event)**

parameters	event	Contains the terminal sending the event and error information if the request failed.

Table 56: Method ApplicationInformationCompleted

3.5.3 BalanceCompleted

The *BalanceCompleted* method is called if an BalanceAsync request finished. If not implemented invokes RequestCompleted.

void **BalanceCompleted(** TimEvent **event**, BalanceResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains counters and print information for the merchant.

Table 57: Method BalanceCompleted

3.5.4 ChangeSettingsCompleted

The *ChangeSettingsCompleted* method is called if an ChangeSettingsAsync request finished. If not implemented invokes RequestCompleted.

void **ChangeSettingsCompleted(** TimEvent **event)**

parameters	event	Contains the terminal sending the event and error information if the request failed.

Table 58: Method ChangeSettingsCompleted

3.5.5 CommitCompleted

The *CommitCompleted* method is called if an CommitAsync request finished. If not implemented invokes RequestCompleted.

void **CommitCompleted(** TimEvent **event)**

parameters	event	Contains the terminal sending the event and error information if the request failed.

Table 59: Method CommitCompleted

3.5.6 CounterRequestCompleted

The *CounterRequestCompleted* method is called if an CounterRequestAsync request finished. If not implemented invokes RequestCompleted.

void **CounterRequestCompleted(** TimEvent **event**, Counters **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains counters.

Table 60: Method CounterRequestCompleted

3.5.7 DeactivateCompleted

The *DeactivateCompleted* method is called if an DeactivateAsync request finished. If not implemented invokes RequestCompleted.

void **DeactivateCompleted(** TimEvent **event**, DeactivateResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains counters and print information for the merchant.

Table 61: Method DeactivateCompleted

3.5.8 DccRatesCompleted

The *DccRatesCompleted* method is called if an DccRatesAsync request finished. If not implemented invokes RequestCompleted.

void **DccRatesCompleted(** TimEvent **event**, PrintData **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains print information for merchant.

Table 62: Method DccRatesCompleted

3.5.9 HardwareInformationCompleted

The *HardwareInformationCompleted* method is called if an HardwareInformationAsync request finished. If not implemented invokes RequestCompleted.

void **HardwareInformationCompleted(** TimEvent **event**, HardwareInformationResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains information about EFT Terminal hardware and EMV kernels and configuration parameters supported by the EFT Terminal.

Table 63: Method HardwareInformationCompleted

3.5.10 InitTransactionCompleted

The *InitTransactionCompleted* method is called if an InitTransactionAsync request finished. If not implemented invokes RequestCompleted.

void **InitTransactionCompleted(** TimEvent **event)**

parameters	event	Contains the terminal sending the event and error information if the request failed.

Table 64: Method InitTransactionCompleted

3.5.11 LoginCompleted

The *LoginCompleted* method is called if an LoginAsync request finished. If not implemented invokes RequestCompleted.

Note: The brands, features, configuration data and terminal identifier in the terminal instance are updated before the first notifier is called. Use event.getTerminal() to retrieve them.

void **LoginCompleted(** TimEvent **event)**

parameters	event	Contains the terminal sending the event and error information if the request failed.

Table 65: Method LoginCompleted

3.5.12 LogoutCompleted

The *LogoutCompleted* method is called if an LogoutAsync request finished. If not implemented invokes RequestCompleted.

void **LogoutCompleted(** TimEvent **event)**

parameters	event	Contains the terminal sending the event and error information if the request failed.

Table 66: Method LogoutCompleted

3.5.13 RebootCompleted

The *RebootCompleted* method is called if an RebootAsync request finished. If not implemented invokes RequestCompleted.

void **RebootCompleted(** TimEvent **event)**

parameters	event	Contains the terminal sending the event and error information if the request failed.

Table 67: Method RebootCompleted

3.5.14 ReconciliationCompleted

The *ReconciliationCompleted* method is called if an ReconciliationAsync request finished. If not implemented invokes RequestCompleted.

void **ReconciliationCompleted(** TimEvent **event**, ReconciliationResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains counters and print information for the merchant.

Table 68: Method ReconciliationCompleted

3.5.15 ReceiptRequestCompleted

The *ReceiptRequestCompleted* method is called if an ReceiptRequestAsync request finished. If not implemented invokes RequestCompleted.

void **ReceiptRequestCompleted(** TimEvent **event**, ReceiptRequestResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains all requested receipts up to a maximum number of receipts the EFT Terminal can send. if HasMoreReceipts is True not all receipts could be send by the EFT Terminal. call ReceiptRequest again to obtain more receipts.

Table 69: Method ReceiptRequestCompleted

3.5.16 ReconfigCompleted

The *ReconfigCompleted* method is called if an ReconfigAsync request finished. If not implemented invokes RequestCompleted.

void **ReconfigCompleted(** TimEvent **event**, PrintData **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains print information for the merchant.

Table 70: Method ReconfigCompleted

3.5.17 RollbackCompleted

The *RollbackCompleted* method is called if an RollbackAsync request finished. If not implemented invokes RequestCompleted.

void **RollbackCompleted(** TimEvent **event**, PrintData **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains print information for the merchant and cardholder.

Table 71: Method RollbackCompleted

3.5.18 SoftwareUpdateCompleted

The *SoftwareUpdateCompleted* method is called if an SoftwareUpdateAsync request finished. If not implemented invokes RequestCompleted.

void **SoftwareUpdateCompleted(** TimEvent **event**, int **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains update status.

Table 72: Method SoftwareUpdateCompleted

3.5.19 SystemInformationCompleted

The *SystemInformationCompleted* method is called if an SystemInformationAsync request finished. If not implemented invokes RequestCompleted.

void **SystemInformationCompleted(** TimEvent **event**, SystemInformationResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains configuration and network information of the terminal.

Table 73: Method SystemInformationCompleted

3.5.20 TransactionCompleted

The *TransactionCompleted* method is called if an TransactionAsync request finished. If not implemented invokes RequestCompleted.

void **TransactionCompleted(** TimEvent **event**, TransactionResponse **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains transaction result information and print information for the merchant and cardholder.

Table 74: Method TransactionCompleted

3.5.21 RequestCompleted

The *RequestCompleted* method is called by all of the other request specific methods unless they are implemented differently.

void **RequestCompleted(** TimEvent **event**, *Object* **data)**

parameters	event	Contains the terminal sending the event and error information if the request failed.
	data	Contains additional request specific data. Can be None-Value if the request in question does not support any additional data. Cast to the matching type to use. Use event.getRequestType() to determine the type of request.

Table 75: Method RequestCompleted

3.5.22 TerminalStatusChanged

The *TerminalStatusChanged* method is called if the state of the terminal changed. Retrieve the state using event.getTerminal().getTerminalState().

void **TerminalStatusChanged(** Terminal **terminal)**

parameters	Terminal	Terminal instance sending the notification.

Table 76: Method TerminalStatusChanged

3.5.23 PrintReceipts

The *PrintReceipts* method is called by all request specific methods containing print data. Convenience method to print all receipts send by the terminal. Contains final receipts to print. If the print data sent to the request completed methods contains fields only ReceiptFormatter is used to create final receipts for printing.

The following request completed methods contain receipts send to this method:

- ActivateCompleted
- BalanceCompleted
- DeactivateCompleted
- DccRatesCompleted
- ReconciliationCompleted
- ReceiptRequestCompleted
- ReconfigCompleted
- RollbackCompleted
- TransactionCompleted

PrintReceipts is called before RequestCompleted.

void **PrintReceipts(** Terminal **terminal**, PrintData **printData)**

parameters	Terminal	Terminal instance receiving the receipt.
	PrintData	Print data containing receipts to print.

Table 77: Method PrintReceipts

3.6 Container Classes

The following classes are containers holding data items for use with methods.

3.6.1 ActivateResponse

The **immutable** *ActivateResponse* class contains the result of calling the Activate or ActivateAsync method. The class contains the following members:

- **immutable** PrintData **PrintData**
- **immutable** int **ActSeqCounter**
  Activation sequence number

3.6.2 Amount

The **immutable** *Amount* class contains the amount for transactions. The amount can be specified in minor units as integer value or in major units as floating point value. The class contains the following members:

- **immutable** int **Value**
  Amount in minor units as integer value.
- **immutable** double **DecimalValue**
  Amount in major units as floating point value.
- **immutable** Currency **Currency**
  Currency to use for transaction. Use a currency constant directly (Currency.CHF) or by the currency code (Currency.withCode("CHF"))
- **constructor Amount(** int **minorUnits**, Currency **currency)**
  Create instance of *Amount* using minor units
- **constructor Amount(** double **majorUnits**, Currency **currency)**
  Create instance of *Amount* using major units

3.6.3 AmountDcc

The **immutable** *AmountDcc* class contains the dcc amount of a transaction. The amount can be specified in minor units as integer value or in major units as floating point value. The class contains the following members:

- **immutable** int **Value**
  Amount in minor units as integer value.
- **immutable** double **DecimalValue**
  Amount in major units as floating point value.
- **immutable** Currency **Currency**
  Currency to use for transaction.
- **immutable** int **Rate**
  Exchange Rate in minor units as integer value
- **immutable** int **RateExponent**
  Exchange Rate Exponent
- **immutable** double **RateDecimal**
  Exchange rate in major units as floating point value
- **immutable** int **Markup**
  Markup applied to the DCC rate by the DCC provider as minor units integer value
- **immutable** int **MarkupExponent**
  Exponent of the markup
- **immutable** double **MarkupDecimal**
  Markup applied to the DCC rate by the DCC provider as major units floating point value

---

3.6.4 Application

The **immutable** *Application* class contains application information. The class contains the following members:

- **immutable** String **Aid**
  Acquirer identifier. Uniquely identifies the acquirer
- **immutable** String **Label**
  Contains the application label of the currently used application

---

3.6.5 Brands

The **immutable** *Brands* class contains information about a brand available on the terminal. It is available after the Login operation has been performed or brands have been manually retrieved by ApplicationInformation or ApplicationInformationAsync. The class contains the following members:

- **immutable** String **Name**
  Contains the brand name of a card
- **immutable** boolean **DccAvailable**
  Specifies if DCC is available for this brand
- **immutable** String **PaymentProtocol**
  Specifies the payment protocol used
- **immutable** int **AcqId**
  Acquirer identifier. Uniquely identifies the acquirer
- **immutable** Date **LastInitDate**
  Last time a defined acquirer was successfully initialized on the terminal
- **immutable** List<Application> **Applications**
  List of applications supported by the brand
- **immutable** List<CurrencyItem> **Currencies**
  List of currencies supported by the brand.

3.6.6 CurrencyItem

The **immutable** *CurrencyItem* class contains information about a currency supported by a brand. The class contains the following members:

- **immutable** Currency **Currency**
  Specifies the supported or used currency. Use a currency constant directly (Currency.CHF) or by the currency code (Currency.withCode("CHF"))
- **immutable** CurrencyType **Type**
  Specifies the type of currency

3.6.7 BalanceResponse

The **immutable** *BalanceResponse* class contains the result of a call to the Balance or BalanceAsync method. The class contains the following members:

- **immutable** PrintData **PrintData**
  Print information for merchant receipt
- **immutable** Counters **Counters**
  Balance counters

3.6.8 ConfigData

The **immutable** *ConfigData* class contains terminal information required for ECR applications to produce receipts on their own. This data is available after the Login operation has been performed. The class contains the following members:

- **immutable** List<String> **ReceiptHeader**
  List of text lines forming the receipt header.
  Remark: Configured in the service center or ECR. Can be overwritten by the card.

- **immutable** String **Language**
  Language to use for printing receipts.

3.6.9 Counter

The **immutable** *Counter* class contains information about a counter for a brand. The class contains the following members:

- **immutable** String **BrandName**
  Name of brand the counter contains information for.

- **immutable** PaymentProtocol **PaymentProtocol**
  Payment protocol linked to the brand.

- **immutable** int **AcqId**
  Acquirer identifier. Uniquely identifies the acquirer

- **immutable** int **Count**
  Total number of transactions.

- **immutable** int **CountDcc**
  Number of DCC related transactions.

- **immutable** int **CountForeign**
  Number of foreign currency related transactions.

- **immutable** List<Total> **Totals**
  Number of totals to break down the counter amount in more detail.

3.6.10 Counters

The **immutable** *Counters* class contains all counters for a specific counter type. Instances of this class are returned by various calls:

- Balance or BalanceAsync
- CounterRequest or CounterRequestAsync
- Deactivate or DeactivateAsync
- Reconciliation or ReconciliationAsync

The class contains the following members:

- **immutable** CounterType **CounterType**
  Specifies what kind of counter are demanded.

- **immutable** int **ActSeqCounter**
  Activation sequence counter.

- **immutable** List<Counter> **Counters**
  Counters per brand.

3.6.11 CardData

The **immutable** *CardData* class contains information about the used payment card. The class contains the following members:

- **immutable** PosEntryMode **PosEntryMode**
  POS entry mode
- **immutable** String **Aid**
  Application identifier
- **immutable** String **Acc**
  Application currency code
- **immutable** String **CardNumber**
  Card number. Only for Non-PCI applications and if available to the terminal.
- **immutable** String **CardNumberPrintable**
  Card number that should be printed on a merchant receipt
- **immutable** String **CardNumberPrintableCardholder**
  Card number that should be printed on a cardholder receipt
- **immutable** String **CardNumberEnc**
  Encrypted card number
- **immutable** String **CardNumberEncKeyIndex**
  Card number encryption key index
- **immutable** Date **CardExpiryDate**
  Card expiration date. Only for Non-PCI applications and if available to the terminal.
- **immutable** String **BrandName**
  Card brand name
- **immutable** String **TenderName**
  Card tender name
- **immutable** List<CardTrackData> **CardTrackDatas**
  List of card track data if present.
- **immutable** List<LoyaltyInformation> **LoyaltyInformations**
  Loyalty information if present

3.6.12 CardTrackData

The **immutable** *CardTrackData* class contains data of a single track. The class contains the following members:

- **immutable** int **TrackNum**
  Specifies the card track (1, 2 or 3)

- **immutable** String **Data**
  Track data as binary string

3.6.13 DeactivateResponse

The **immutable** *DeactivateResponse* class is the result of calling the Deactivate or DeactivateAsync method. The class contains the following members:

- **immutable** PrintData **PrintData**
  Print data for merchant receipt.

- **immutable** Counters **Counters**
  Shift counters.

3.6.14 DisplayLine

The **immutable** *DisplayLine* class contains a single line of text displayed on the EFT Terminal. The class contains the following members:

- **immutable** int **Line**
  Line number starting with 1 for the first line at the top

- **immutable** String **Text**
  Text on the line

3.6.15 EcrInfo

The *EcrInfo* class defines one ECR information token for use by the SoftwareInformation or SoftwareInformationAsync method. The class contains the following members:

- String **Type**
  e.g. *EcrApplication*

- String **Name**
  e.g. *proPOS*

- String **ManufacturerName**

- String **Version**
  e.g. *V2016.02.01*

- String **SerialNumber**
  e.g. *4711-0815*

- String **Architecture**
  e.g. *Mobile*

3.6.16 Features

the **immutable** *Features* class contains features supported by the terminal. The class contains the following members:

- **immutable** int **ProtocolLevel**
  Supported protocol level. The value is the supported ProtocolLevel number. Present once per supported protocol level.

- **immutable** EnumSet<Guides> **Guides**
  Specifies which SIXml Guides are supported by the EFT Terminal.

- **immutable** bool **AutoCommit**
  Auto-commit support by the terminal. If false user has to call Commit or CommitAsync unless auto-commit is enabled on the TIMApi level.

3.6.17 GlobalSettings

The *GlobalSettings* class contains global settings affecting all *Terminal* instances unless overridden by TerminalSettings. The class contains the following members:

- String **LogDir**
  Defines where the Lof File has to be generated

Alternatively, the GlobalSettings can be defined in a configuration file, however it's important to note that the Settings from the API have a higher priority.

3.6.18 Hardware

The **immutable** *Hardware* class contains information about a piece of hardware in the terminal or devices attached to it. The class contains the following members:

- **immutable** HardwareType **HardwareType**
  Specifies the hardware.
- **immutable** String **SerialNumber**
  Serial number of the specified hardware.
- **immutable** Date **ProductionDate**
  Production date of the specified hardware.
- **immutable** String **ProductVersion**
  Product version of the specified hardware.
- **immutable** String **FirmwareVersion**
  Firmware version of the specified hardware.
- **immutable** SecurityStatus **SecurityStatus**
  Security status of the specified hardware.
- **immutable** String **RemoteIp**
  IP address of the device on which it is accessible from external sources.
- **immutable** Date **LastCleaningDate**
  Available if HardwareType is *ContactReader* or *MagStripeReader*

3.6.19 HardwareInformationResponse

The **immutable** *HardwareInformationResponse* class contains the result of calling the HardwareInformation or HardwareInformationAsync emthod. The class contains the following members:

- **immutable** List<Hardware> **Hardwares**
  List of hardware attached to the EFT Terminal
- **immutable** List<KernelVersion> **KernelVersions**
  Kernel versions supported by the EFT Terminal
- **immutable** List<Setting> **Settings**
  Settings supported by the EFT Terminal

3.6.20 KernelVersion

The **immutable** *KernelVersion* class contains information about a kernel version supported by the EFT Terminal. The class contains the following members:

- **immutable** KernelType **KernelType**
  Type of kernel
- **immutable** String **Version**
  Version string of kernel

3.6.21 LoginRequest

The *LoginRequest* class contains explicit login information for use with Login or LoginAsync methods. Without using *LoginRequest* the information is assembled from the default parameters stored in the terminal properties. The class contains the following members:

- String **PosId**
  POS identifier. Overrides values set by the configuration file or using *SetPosId* method in the *Terminal* instances.
- String **IntegratorId**
  Integrator identifer.
- List<PrintOption> **PrintOptions**
  Print options overriding the print options set by the configuration file or using *SetPrintOptions* method in the *Terminal* instances.
- int **ManufacturerFlags**
  Manufacturer flags overriding the print options set by the configuration file or using *SetManufacturerFlags* in the *Terminal* instances.

3.6.22 LoyaltyInformation

The **immutable** *LoyaltyInformation* class containing loyalty information for a specific type. The class contains the following members:

- **immutable** String **LoyaltyInfoType**
  Type of loyalty data
- **immutable** String **Value**
  Loyalty information data

3.6.23 MerchantOption

The **immutable** *MerchantOption* class containing merchant specific options. The class contains the following members:

- **immutable** MerchantOptionType **Type**
  Type of merchant option

- **immutable** String **Value**
  Merchant option data

3.6.24 NetworkInformation

The **immutable** *NetworkInformation* class contains network information of the terminal. Result of calling SystemInformation or SystemInformationAsync. The class contains the following members:

- **immutable** String **TerminalIp**
  Terminal IP

- **immutable** String **TerminalIpMask**
  Terminal IP network mask

- **immutable** String **TerminalIpGw**
  Terminal IP gatewat

- **immutable** String **TerminalIpDns**
  Terminal IP dns host name

3.6.25 PrintData

The **immutable** *PrintData* class contains receipts or receipt items for printing by the ECR depending on the PrintOptions used during login.

- **immutable** List<Receipt> **Receipts**
  Receipts to print by the ECR. Present if print type is *Normal*

- **immutable** List<ReceiptItems> **ReceiptItems**
  Receipt items for the ECR to create receipts for printing. Present if print type is *FieldsOnly*

3.6.26 PrintOption

The *PrintOption* class define print options for a specific receipt type. Used for Login or LoginAsync calls. The class contains the following members:

- Recipient **Recipient**
  Target of the print option. Can be *Merchant* or *Cardholder*
- PrintFormat **PrintFormat**
  Specifies the print format.
- int **PrintWidth**
  Specifies the print width of the receipt (**default** = 40)
- EnumSet<PrintFlags> **PrintFlags**
  Specifies receipt formatting options.

3.6.27 Receipt

The **immutable** *Receipt* class contains a receipt to be printed by the ECR. The class contains the following members:

- **immutable** Recipient **Recipient**
  Recipient of the receipt.
- **immutable** String **Value**
  Receipt to print

3.6.28 ReceiptItem

The **immutable** *ReceiptItem* class contains the value of a specific receipt item to be used by the ECR to create receipts. The class contains the following members:

- **immutable** ReceiptItemType **ReceiptItemType**
  Type of receipt item.
- **immutable** String **Value**
  Value of the receipt item

3.6.29 ReceiptItems

The **immutable** *ReceiptItems* class contains a list of ReceiptItem to be used for a specific type of receipts. The class contains the following members:

- **immutable** ReceiptType **ReceiptType**
  Type of the receipt.
- **immutable** List<ReceiptItem> **ReceiptItem**
  List of receipt items to be used for this receipt type.

3.6.30 ReceiptRequestResponse

The **immutable** *ReceiptRequestResponse* class contains the result of a call to the ReceiptRequest or ReceiptRequestAsync method. The class contains the following members:

- **immutable** PrintData **PrintData**
  Print information for receipts to print by the ECR.

- **immutable** boolean **HasMoreReceipts**
  States if the maximum number of receipts has been transmitted and more receipts can be requested

---

3.6.31 ReconciliationResponse

The **immutable** *ReconciliationResponse* class contains the result of calling the Reconciliation or ReconciliationAsync method. The class contains the following members:

- **immutable** Counters
  Balance counters

- **immutable** PrintData
  Print information for merchant receipt

3.6.32 Setting

The *Setting* class contains a setting to change on the EFT terminal. Used for the ChangeSettings or ChangeSettingsAsync method. The class contains the following members:

- SettingType **SettingType**
  Setting type to modify.

- String **Value**
  Value to set for setting

3.6.33 SignatureInformation

The **immutable** *SignatureInformation* contains signature information captured by the EFT Terminal as a result to calling the Transaction or TransactionAsync method. The class contains the following members:

- **immutable** ImageFileFormat **ImageFileFormat**
  Image file format
- **immutable** int **ImageWidth**
  Image width in pixels
- **immutable** int **ImageHeight**
  Image height in pixels

3.6.34 SystemInformationResponse

The **immutable** *SystemInformationResponse* class contains the result of calling the SystemInformation or SystemInformationAsync method. The class contains the following members:

- **immutable** NetworkInformation **tNetworkInformation**
  Network information of EFT terminal

3.6.35 TerminalStatus

The **immutable** *TerminalStatus* class contains information about the current EFT Terminal status. The class contains the following members:

- **immutable** List<DisplayLine> **DisplayContent**
  Content of display on EFT Terminal.
- **immutable** ConnectionStatus **ConnectionStatus**
  Connection status of the EFT Terminal.
- **immutable** ManagementStatus **ManagementStatus**
  Management status of the EFT Terminal.
- **immutable** CardReaderStatus **CardReaderStatus**
  Status of card reader attached to EFT Terminal.
- **immutable** SleepModeStatus **SleepModeStatus**
  Sleep mode status of EFT Terminal.
- **immutable** boolean **ReceiptInformation**
  Receipts can be retrieved using ReceiptRequest or ReceiptRequestAsync
- **immutable** CardData **CardData**
  Information about payment card used by the customer if present

---

3.6.36 TimEvent

The **immutable** *TimEvent* class contains event information for asynchronous method calls. It is send to notifiers. The class contains the following members:

- **immutable** Terminal **Terminal**
  Terminal instance sending event
- **immutable** TimException **TimException**
  None if request finished successfully or an TimException instance if the request failed
- **immutable** RequestType **RequestType**
  Type of request this event has been send for.

3.6.37 TimException

The **immutable** *TimException* class contains information about a failed request or method call. For synchronous method calls *TimException* is thrown as exception. For asynchronous method calls *TimException* is included in the TimEvent. The class contains the following members:

- **immutable** ResultCode **ResultCode**
  Contains the result code with the error
- **immutable** String **ErrorMessage**
  Error message in the ECR language
- **immutable** String **NativeError**
  Native error if present

3.6.38 Total

The **immutable** *Total* class contains information about the total for one currency. The class contains the following members:

- **immutable** int **Count**
  Number of transactions.
- **immutable** Amount **AmountSum**
  Total amount per currency.
- **immutable** List<TrxDetail> **TrxDetails**
  List of transaction details.

3.6.39 TransactionData

The *TransactionData* contains configuration for transaction used by **Transaction** or **TransactionAsync** method. The class contains the following members:

- boolean **DccAllowed**
  Allows the EFT Terminal to enable DCC function
- boolean **PartialApprovalAllowed**
  Allows the EFT Terminal to enable partial approval for the transaction.
- Date **TrxOriginalDate**
  Timestamp of the original transaction
- int **EcrSeqCounter**
  ECR sequence counter
- String **TransRef**
  Transaction reference defined by the terminal
- String **TransSeq**
  Transaction sequence number defined by the terminal

3.6.40 TransactionInformation

The **immutable** *TransactionInformation* class contains information about a finished transaction after calling the **Transaction** or **TransactionAsync** method. The class contains the following members:

- **immutable** PosEntryMode **PosEntryMode**
  According to EMV definition
- **immutable** Cvm **Cvm**
  Cardholder verification method
- **immutable** MerchantAction **MerchantAction**
  Feedback to know which merchant action has to be performed
- **immutable** String **AuthCode**
  Authorization code received from the acquirer
- **immutable** DateTime **TimeStamp**
  Local time of the transaction
- **immutable** String **TransRef**
  Transaction reference defined by the terminal
- **immutable** int **AcqId**
  Acquirer identifier
- **immutable** String **Aid**
  Application identifier

3.6.41 TransactionRequest

The *TransactionRequest* contains configuration for transaction used by **Transaction** or **TransactionAsync** method. The class contains the following members:

- String **UserId**
  ECR user identifier
- Amount **Amount**
  Transaction amount and currency
- TransactionData **TransactionData**
  Transaction information
- List<MerchantOption> **MerchantOptions**
  Additional merchant options

3.6.42 TransactionResponse

The **immutable** *TransactionResponse* class contains the result of a transaction finished after calling the **Transaction** or **TransactionAsync** method. The class contains the following members:

- **immutable** TransactionType **TransactionType**
  Type of transaction
- **immutable** Amount **Amount**
  Amount authorized by the transaction
- **immutable** Amount **AmountDue**
  Due amount in the transaction
- **immutable** AmountDcc **AmountDcc**
  DC amount authorized by the transaction if present
- **immutable** Amount **AmountSaldo**
  Saldo amount if present
- **immutable** TransactionInformation **TransactionInformation**
  Information about completed transaction
- **immutable** SignatureInformation **SignatureInformation**
  Signature captured by EFT Terminal if present
- **immutable** String **DccDisclaimer**
  Disclaimer sent by host
- **immutable** CardData **CardData**
  Information about payment card used by the cardholder if present
- **immutable** PrintData **PrintData**
  Print information for merchant and cardholder receipts

3.6.43 TrxDetail

The **immutable** *TrxDetail* class contains transaction details for counters. The class contains the following members:

- **immutable** boolean **DccFlag**
  Specifies is a transaction is a DCC transaction or not
- **immutable** TransactionType **TransactionType**
  Transaction type this details affect
- **immutable** int **Count**
  Number of transactions covered by the sum
- **immutable** Amount **AmountSum**
  Total of amount

3.7 Enumerations

The following sections show the enumerations that can be used in the TIM API. *ResultCodes* are enumerations as well but are specified in their own chapter after this one.

3.7.1 CardReaderStatus

The *CardReaderStatus* enumeration defines the possible card reader states of the EFT Terminal attached card reader. The following values are allowed:

Name	Value	Remark
CardReaderClosed	CardReaderClosed	The shutter of the card reader is closed. No cards can be inserted.
CardManuallyEntered	CardManuallyEntered	Card information has been manually entered. Valid for Manual-Pan-Key.
CardSwiped	CardSwiped	Card has been swiped. Valid for Mag-Stripe.
CardNotRemoved	CardNotRemoved	Card has not been removed by the cardholder despite being prompted to do so (timeout: 15s). Valid for ICC.
CardPresented	CardPresented	Contactless card has been presented. Only for ICC contactless transactions. Valid for Contactless.
CardReaderEmpty	CardReaderEmpty	The card reader is empty and accepts cards.
CardInserted	CardInserted	Card in the chip reader.
CardEjected	CardEjected	Card has been ejected by the application.

Table 78: Enumeration CardReaderStatus

3.7.2 ConnectionMode

The *ConnectionMode* enumeration defines the type of connection between ECR and EFT Terminal. The following values are allowed:

Name	Value	Remark
Broadcast	Broadcast	ECR and EFT Terminal exchange connection parameters using broadcasting before connecting.
OnFixIP	OnFixIP	ECR and EFT Terminal connect directly using pre-configured connection parameters.

Table 79: Enumeration ConnectionMode

3.7.3 ConnectionStatus

The *ConnectionStatus* enumeration defines the possible connection states of the EFT Terminal. The following values are allowed:

Name	Value	Remark
Disconnected	Disconnected	Disconnected
LoggedOut	LoggedOut	Logged out
LoggedIn	LoggedIn	Logged in

Table 80: Enumeration ConnectionStatus

3.7.4 CurrencyType

The *CurrencyType* enumeration defines the type of currency. The following values are allowed:

Name	Value	Remark
Local	Local	Local currency depending on location
Foreign	Foreign	Foreign currency
Dcc	Dcc	Currency used for DCC conversion

Table 81: Enumeration CurrencyType

3.7.5 CounterType

The *CounterType* enumeration defines the possible counter types for CounterRequest and CounterRequestAsync method. The following values are allowed:

Name	Value	Remark
Balance	Balance	The *Balance*-type to get the counters since the last Balance or BalanceAsync method has been called.
Shift	Shift	The *Shift*-type is to get the counters since the last Activate or ActivateAsync method has been called.

Table 82: Enumeration CounterType

3.7.6 Currency

The *Currency* enumeration defines the possible currencies. These are normed currency codes like *CHF* or *EUR* as defined in [B7] ISO-4217. Currency enumeration values can be used directly like *Currency.CHF* or using the method

- Currency **WithCode(** String **currencyCode)**

Currency entries contain properties with additional information:

- **immutable** int **Code**
  Normed currency code.
- **immutable** int **Exponent**
  Currency major units exponent.
- **immutable** Map<String,String> **Names**
  Map of display names keyed by language.

For convenience the following methods exist:

- double **Decimal(** int **amount)**
  Convert amount in minor units to amount in major units.
- String **Name(** String **language)**
  Display name of currency for language. Uses *Names* property. If no name entry exists in the property the *Code* property is returned instead.

Where *currencyCode* is the normed 3-letter currency code. Thus *Currency.CHF* is the same as calling *Currency.GetWithCode("CHF")*. This can be used for storing currency as string in the ECR retrieving the correct enumeration type where required.

3.7.7 Cvm

The *Cvm* enumeration defines which cardholder verification method has been performed during the transaction.

Name	Value	Remark
Pin	Pin	PIN verification
PinSignature	PinSignature	PIN verification and signature (paper)
Signature	Signature	Signature (paper)
NoCvm	NoCvm	No CVM required

Table 83: Enumeration Cvm

3.7.8 Guides

The *Guides* enumeration contains flags for specifications supported on the EFT terminal or requested by the ECR. The following values are allowed:

Name	Value	Remark
Retail	0h1	Retail & Mobile. Fundamental retail business cases.
Unattended	0h2	Unattended Terminals in vending machines.
Advanced Retail	0h4	Advanced Retail additions: TopUp, Purchase with Cashback etc.
Banking	0h8	Banking business cases.
Petrol	0h10	Pre-Authorization and fleet management for petrol business.
Dialog	0h20	Dialog mode addons.
Remote	0h40	Remote addons.
Gastro	0h80	Addons for restaurants.
Hospitaliy	0h100	Addons for restaurants

Table 84: Enumeration Guides

3.7.9 HardwareType

The *HardwareType* enumeration defines the type of hardware attached to the EFT terminal. The following values are allowed:

Name	Value	Remark
Terminal	Terminal	Terminal device related, not the secure part.
EftApplication	EftApplication	Software that runs on the terminal.
PinPad	PinPad	Secure part of the terminal.
ContactReader	ContactReader	Contact ICC reader.
ContactlessReader	ContactlessReader	Contactless card reader.
MagStripeReader	MagStripeReader	Magnetic stripe reader.

Table 85: Enumeration HardwareType

3.7.10 ImageFileFormat

The *ImageFileFormat* enumeration defines the possible image file formats of signature images. The following values are allowed:

Name	Value	Remark
Png	Png	Png
Jpeg	Jpeg	Jpeg

Table 86: Enumeration ImageFileFormat

3.7.11 KernelType

The *KernelType* enumeration defines the possible types of supported kernels. The following values are allowed:

Name	Value	Remark
EmvContact	-1	EMV Contact kernel.
EntryPoint	0	EMV Contactless Entrypoint.
Kernel1	1	EMV Contactless kernel number 1 according to kernel numbering as defined by EMV contactless [B4].
Kernel2	2	EMV Contactless kernel number 2 according to kernel numbering as defined by EMV contactless [B4].
Kernel3	3	EMV Contactless kernel number 3 according to kernel numbering as defined by EMV contactless [B4].
Kernel4	4	EMV Contactless kernel number 4 according to kernel numbering as defined by EMV contactless [B4].
Kernel5	5	EMV Contactless kernel number 5 according to kernel numbering as defined by EMV contactless [B4].
Kernel6	6	EMV Contactless kernel number 6 according to kernel numbering as defined by EMV contactless [B4].
Kernel7	7	EMV Contactless kernel number 7 according to kernel numbering as defined by EMV contactless [B4].

Table 87: Enumeration KernelType

3.7.12 ManagementStatus

The *ManagementStatus* enumeration defines the possible management states of the EFT Terminal. The following values are allowed:

Name	Value	Remark
Closed	Closed	Initial state, shift closed. In this state only administration, remote and status functions can be used, which means no interaction with the cardholder is required.
Open	Open	Shift opened. In this state only financial, non-financial, remote and status functions as well as Deactivate can be used. Usually an interaction with the cardholder is required.
Dialog	Dialog	Dialog mode opened. This state allows to process ECR controlled dialog functions. Only dialog and remote functions can be used. Supported only in some guides

Table 88: Enumeration ManagementStatus

3.7.13 MerchantAction

The *MerchantAction* enumeration defines the possible actions taken by the merchant during CVM. The following values are allowed:

Name	Value	Remark
Signature	Signature	The merchant has to sign the receipt.
None	None	No merchant action is required

Table 89: Enumeration MerchantAction

3.7.14 MerchantOptionType

The *MerchantOptionType* enumeration defines the possible merchant options. The following values are allowed:

Name	Value	Remark
AdditionalMerchantData	AdditionalMerchantData	Additional data the merchant would like to add. String value.
MultiAccountIndex	MultiAccountIndex	Account index that shall be used for this transaction. Numeric number that defines the account index.
MultiContractIndex	MultiContractIndex	Acquirer contract index that shall be used for this transaction. Numeric number that specifies the contract index.
MerchantTid	MerchantTid	Specific terminal identifier defined by the merchant.

Table 90: Enumeration MerchantOptionType

3.7.15 PaymentProtocol

The *PaymentProtocol* enumeration defines the support payment protocols. The following values are allowed:

Name	Value	Remark
EP2	ep2	EFTPOS 2000 protocol
EV	ev	EV protocol
ValueMaster	vm	ValueMaster protocol
3CXml	3CXml	3CXml

Table 91: Enumeration PaymentProtocol

3.7.16 PosEntryMode

The *PosEntryMode* enumeration defines the support POS entry modes. The following values are allowed:

Name	Value	Remark
Unspecified	0	Unspecified
Manual	1	Manual
MagStripeIncomplete	2	Magnetic stripe, only partial data available. Track data checking may not possible,minimum requirement: PAN data
BarCode	3	Bar Code
OCR	4	OCR
ICC	5	Integrated circuit card
MagStripe	90	Magnetic stripe read completely. Track data checking is possible, service code does not begin with “2” or “6”
MagStripeFallback	91	Fallback (normal case): magnetic stripe read; service code begins with “2” or “6”. Last transaction in the transaction log was a successful chip or magnetic stripe transaction
MagStripeFallbackAgain	92	Fallback: magnetic stripe read; service code begins with “2” or “6”. Last transaction in the transaction log was also a fallback transaction. If the number of transactions with PosEntryMode=“92” grows, the chip reader could be out of order
MagStripeFallbackIccFail	93	Fallback: magnetic stripe read; service code begins with “2” or “6”; transaction processing with IC failed. Code “93” is used at the transaction, which has caused the fallback, if the chip has been contacted, an application has been selected and the chip data could be read (trm-func, steps 11.4.5, 11.4.8 and 11.4.9), but the following processing with the chip has been aborted because of unknown reasons
EmergencyDataEntry	94	Emergency data entry: The emergency receipt data is entered at acquirer
ECommerce	95	E-Commerce PSP transaction (distance payment)
CtlessIcc	97	Contactless Chip (EMV)
CtlessMagStrige	98	Contactless Magstripe

Table 92: Enumeration PosEntryMode

3.7.17 ProtocolType

The *ProtocolType* defines the protocol used between ECR and EFT Terminal. The following values are allowed:

Name	Value	Remark
SIXml	2	Use SIXml protocol

Table 93: Enumeration ProtocolType

3.7.18 PrintFlags

The *PrintFlags* enumeration contains flags for receipt formatting options. The following values are allowed:

Name	Value	Remark
SuppressHeader	SuppressHeader	Do not print receipt headers
SuppressSignature	SuppressSignature	Do not print signature line
SuppressEcrInfo	SuppressEcrInfo	Do not print ECR information
SuppressEftInfo	SuppressEftInfo	Do not print EFT terminal information

Table 94: Enumeration PrintFlags

3.7.19 PrintFormat

The *PrintFormat* enumeration defines the possible print formats supported by printers. The following values are allowed:

Name	Value	Remark
NoPrint	NoPrint	Do not generate any receipts
Normal	Normal	Receipts are generated, formatted and sent to the ECR. **(default)**
OnDevice	OnDevice	Receipts are generated and printed on the printer attached to the EFT terminal
FieldsOnly	FieldsOnly	Only fields are returned for printing, not a generated receipt

Table 95: Enumeration PrintFormat

3.7.20 ReceiptItemType

The *ReceiptItemType* enumeration defines the possible types of item information for receipts. The following values are allowed:

Name	Value	Remark
ActId	ActId	Activation Id
AccPer	AccPer	Accounting Period
AcqId	AcqId	AcquirerId
Aid	Aid	Application identifier
Amount	Amount	Amount of the transaction
AmountDcc	AmountDcc	DCC amount of the transaction
AuthCode	AuthCode	Authorization code
BrandName	BrandName	BrandName of the card used
Currency	Currency	Currency of the transaction
CurrencyDcc	CurrencyDcc	DCC currency of the transaction
DccDisclaimer	DccDisclaimer	DCC disclaimer
Disclaimer	Disclaimer	Standard disclaimer
Exponent	Exponent	Exponent of the currency used
ExponentDcc	ExponentDcc	Exponenet of the DCC currency
MarkupDcc	MarkupDcc	DCC markup
MarkupExponentDcc	MarkupExponentDcc	Exponent of DCC markup
CardNumberPrintableMerchant	CardNumberPrintableMerchant	Card number for merchant receipt
CardNumberPrintableCardholder	CardNumberPrintableCardholder	Card number for cardholder receipt
RateDcc	RateDcc	DCC rate
RateExponentDcc	RateExponentDcc	Exponent of DCC rate
TimeStamp	TimeStamp	Time stamp of transaction
TrmId	TrmId	Terminal identifier
TrxRefNum	TrxRefNum	Transaction reference number
TrxSeqCnt	TrxSeqCnt	Transaction sequence counter

Table 96: Enumeration ReceiptItemType

3.7.21 ReceiptRequestType

The *ReceiptRequestType* enumeration defines the possible types of receipts to request. The following values are allowed:

Name	Value	Remark
Reprint	Reprint	Only the latest receipt is returned for reprinting.
List	List	A list of collected silent receipts is returned.

Table 97: Enumeration ReceiptRequestType

3.7.22 ReceiptType

The *ReceiptType* enumeration defines the possible types of receipts. The following values are allowed:

Name	Value	Remark
Purchase	Purchase	Purchase receipt
Credit	Credit	Credit receipt
Reversal	Reversal	Reversal receipt

Table 98: Enumeration ReceiptType

3.7.23 Recipient

The *Recipient* enumeration defines the possible recipients of receipts. The following values are allowed:

Name	Value	Remark
Merchant	Merchant	The datum is intended for the merchant
Cardholder	Cardholder	The datum is intended for the cardholder
Both	Both	The datum is intended for the cardholder and the merchant

Table 99: Enumeration Recipient

3.7.24 RequestType

The *RequestType* enumeration defines the request types used in the TIMApi to communicate the result of finished operations to notifiers. The following values are allowed:

Name	Value	Remark
Activate	Language Specific	Operation started by ActivateAsync.
ApplicationInformation	Language Specific	Operation started by ApplicationInformationAsync.
Balance	Language Specific	Operation started by BalanceAsync.
ChangeSettings	Language Specific	Operation started by ChangeSettingsAsync.
Commit	Language Specific	Operation started by CommitAsync.
CounterRequest	Language Specific	Operation started by CounterRequestAsync.
Deactivate	Language Specific	Operation started by DeactivateAsync.
DccRates	Language Specific	Operation started by DccRatesAsync.
HardwareInformation	Language Specific	Operation started by HardwareInformationAsync.
Login	Language Specific	Operation started by LoginAsync.
Logout	Language Specific	Operation started by LogoutAsync.
Reboot	Language Specific	Operation started by RebootAsync.
Reconciliation	Language Specific	Operation started by ReconciliationAsync.
ReceiptRequest	Language Specific	Operation started by ReceiptRequestAsync.
Reconfig	Language Specific	Operation started by ReconfigAsync.
Rollback	Language Specific	Operation started by RollbackAsync.
SoftwareUpdate	Language Specific	Operation started by SoftwareUpdateAsync.
SystemInformation	Language Specific	Operation started by SystemInformationAsync.
Transaction	Language Specific	Operation started by TransactionAsync.

Table 100: Enumeration RequestType

3.7.25 SecurityStatus

The *SecurityStatus* enumeration defines the possible security states of terminal hardware. The following values are allowed:

Name	Value	Remark
Disabled	Disabled	The security has actively been disabled (i.e. some form of test mode)
Active	Active	The security is working correctly.
Tampered	Tampered	The security system is in tampered state. Key lost

Table 101: Enumeration SecurityStatus

3.7.26 SettingType

The *SettingType* enumeration defines the possible types of settings to configure on the EFT Terminal. The following values are allowed:

Name	Value	Remark
DisplayBrightness	DisplayBrightness	Brightness of the display.
DisplayContrast	DisplayContrast	Contrast of the display.
KeypadTones	KeypadTones	Volume of the key press tones.
AlertTones	AlertTones	Volume of the alert tones.
Language	Language	Language of the EFT.

Table 102: Enumeration SettingType

3.7.27 SleepModeStatus

The *SleepModeStatus* enumeration defines the possible sleep mode states of the EFT Terminal attached card reader. The following values are allowed:

Name	Value	Remark
EnteringSleep	EnteringSleep	Terminal enters sleep mode.
WakingUp	WakingUp	Terminal wakes up from sleep mode.

Table 103: Enumeration SleepModeStatus

3.7.28 TransactionStatus

The *TransactionStatus* enumeration defines the possible transaction states of the EFT Terminal attached card reader. The following values are allowed:

Name	Value	Remark
Idle	Idle	No transaction running.
ReadingCard	ReadingCard	Reading card data and matching applications.
ApplicationSelection	ApplicationSelection	Application selection is ongoing. May require cardholder interaction.
WaitForProceed	WaitForProceed	Terminal waits for another ECR command.
DccSelection	DccSelection	Dcc selection in progress.
EnterTip	EnterTip	Tip Entry in progress.
PinEntry	PinEntry	CVM PIN has been selected. PIN entry in progress.
SignatureCapture	SignatureCapture	CVM Signature has been selected. Signature capture ongoing.
NoCVM	NoCVM	Pseudo-state, no CVM is required.
Processing	Processing	Payment processing in progress.
WaitForCommit	WaitForCommit	AutoCommit is disabled. Waiting for Commit from ECR.

Table 104: Enumeration TransactionStatus

3.7.29 TransactionType

The *TransactionType* enumeration defines the possible financial transaction types. The following values are allowed:

Name	Value	Remark
Purchase	Language Specific	The *Purchase*-function is a standard sale transaction.
Credit	Language Specific	The *Credit*-function is a standard refund transaction.
Reversal	Language Specific	The *Reversal*-function voids a previous transaction. The transaction to reverse is selected with the *TransRef*-property

Table 105: Enumeration TransactionType

3.7.30 UpdateStatus

The *UpdateStatus* enumeration defines the current update status of the terminal. The following values are allowed:

Name	Value	Remark
UpToDate	UpToDate	The actual software is up to date and no software update will be made.
RunningNoReboot	RunningNoReboot	The software update is running. No terminal reboot is needed to complete it.
RunningReboot	RunningReboot	The software update is running. At least one (or more) reboot(s) is/are needed to complete it.

Table 106: Enumeration UpdateStatus

3.8 Result Codes

The following result codes are enumerations used by the API to indicate various failure conditions.

Result code enumeration values can be used directly like *ResultCode.OK* or using the method

- ResultCode **WithCode(** int **code)**

Result code entries contain properties with additional information:

- **immutable** int **Code**
  Result code.

- **immutable** Map<String,String> **ErrorMessage**
  Map of error messages keyed by language.

For convenience the following methods exist:

- String **ErrorMessage(** String **language)**
  Error message for language. Uses *ErrorMessages* property. If no error message entry exists in the property the name of the enumeration constant is returned instead.

3.8.1 Positive results

Code	Description
0	Request has been successful

Table 107: Positive results

3.8.2 TIM API related results - General

Code	Error Message	Description
50	Cancelled by ECR	Current request has been cancelled by the ECR.
51	Invalid answer	The ECR received an invalid answer.
52	Message belongs to disabled feature	The current request cannot be handled because the required feature is disabled.
53	Function not allowed - License issue	The requested function is not allowed with the current license Communication.
54		Persistent data damaged, out of space orstorage permission problem.

Table 108: TIM API related results - General

3.8.3 TIM API related results - Communication

Code	Error Message	Description
100	Could not connect to TIM Server	The ECR was not able to connect to the TIM Server.
101	Could not connect to EFT Terminal	The ECR was not able to connect to the EFT Terminal.
102	Connection lost to TIM Server	The ECR lost the connection to the TIM Server.
103	Connection lost to EFT Terminal	The ECR lost the connection to the EFT Terminal.
104	Ethernet cable not connected	No ethernet cable has been connected.
105	RS232 error	No RS232 cable connected or wrong port configured.
106	Timeout - Communication TIM Server	Timeout during connection establishment from TIM API to TIM Server.
107	Timeout - Communication EFT Terminal	Timeout during connection establishment from TIM API to EFT Terminal.

Table 109: TIM API related results - Communication

3.8.4 TIM Server related results - General

Code	Error Message	Description
200	Invalid answer	TIM Server received an invalid answer.
201	Invalid request	TIM Server received an invalid request from TIM API.
202	Message belongs to disabled feature	TIM Server received a message that requires a disabled feature.
203		Persistent data damaged, out of space or storage permission problem.

Table 110: TIM Server related results - General

3.8.5 TIM Server related results - Communication

Code	Error Message	Description
250	Could not connect to EFT Terminal	TIM Server was not able to connect to EFT Terminal.
251	Connection lost to EFT Terminal	TIM Server lost connection to the EFT Terminal.
252	Timeout - Communication EFT Terminal	Timeout during connection establishment from TIM Server to EFT Terminal.

Table 111: TIM Server related results - Communication

3.8.6 TIM API related results - Hardware General

Code	Error Message	Description
300	Device not available - CCR	Chip card reader not available.
301	Device not available - MCR	Magstripe card reader not available.
302	Device not available - NFC	Contactless card reader not available.
303	Device not available - Display	Display not available.
304	Device not available - PinPad	PinPad not available.
305	Device not available - RS232	RS232 device not available.
306	Device not configured - RS232	Connected RS232 device not configured.

Table 112: TIM API related results - Hardware General

3.8.7 TIM API related results - CardReader

Code	Error Message	Description
400	Card reader error - CCR	Chip card reader error occured.
401	Card reader error - MCR	Magstripe card reader error occured.
402	Card reader error - NFC	Contactless card reader error occured.
403	Card error - CCR	Card error occured using the chip card reader.
404	Card error - MCR	Card error occured using the magstripe card reader.
405	Card error - NFC	Card error occured using the contactless card reader.
406	Read error	Reading error occured during card reading.
407	Read timeout	Timeout occured during card reading.
408	Card insertion timeout	Cardholder did not insert card during timeout.
409	Keys lost - Card reader	Keys lost for secure communication.
410	Security error	Security error occured.
411	Card timeout	Card did not respond in time.
412	Card not readable	Card could not be read.
413	Card - invalid data	Invalid card data read.

Table 113: TIM API related results - CardReader

3.8.8 TIM API related results - PinPad

Code	Error Message	Description
500	Security error	Security error occured.
501	Tampered	PinPad has be tampered.
502	Keys lost - PinPad	Keys lost for secure communication.

Table 114: TIM API related results - PinPad

3.8.9 TIM API related results - Cardholder

Code	Error Message	Description
550	Stop - Cardholder	Cardholder cancelled transaction pushing the Stop key.
551	Timeout - Cardholder	Cardholder did not react in time.
552	Card removed	Card has been removed by the cardholder.

Table 115: TIM API related results - Cardholder

3.8.10 TIM API related results - Communication

Code	Error Message	Description
600	Timeout - ECR	ECR did not respond in time.
601	Could not connect to payment host	EFT terminal not able to connect to payment host.
602	Connection lost to payment host	EFT Terminal did lose connection to payment host.
603	Timeout - Answer RS232	RS232 device did not respond in time.
604	Communication failure	Communication failed.

Table 116: TIM API related results - Communication

3.8.11 TIM API related results - SIXml Protocol

Code	Error Message	Description
700	General error	General error on SIXml Protocol.
701	Invalid request - Generic	Generic error if request not valid.
702	Invalid request - Wrong cashier	Wrong UsrId entered for Activate.
703	Invalid request - Wrong EcrId	Wrong EcrId entered for Login.
704	Invalid request - Unknown reference number	Unknown reference number entered for Credit or Reversal.
705	Message not allowed in this state	Request not allowed in the current terminal state.
706	Busy (Other controller)	Another controller is currently using the EFT.
707	Busy (Maintenance)	Mainenance task in progress.
708	Busy - Another request running	Another request from the same controller is already running.

Table 117: TIM API related results - SIXml Protocol

3.8.12 TIM API related results - Transaction

Code	Error Message	Description
800	No common applications found	No common application found on card during Application Selection.
801	Transaction limit exceeded	Current transaction exceeds transaction limit.
802	No common CVM found	No common cardholder verification method found on card during.
803	Declined - CVM failed	Performed CVM failed.
804	Referral	Referral required for current transaction.
805	Invalid auth response	Authorization response invalid.
806	Authorization declined - Generic	Authorization failed, generic error.
807	Authorization declined - Saldo too low	Authorization failed, saldo too low.
808	Authorization declined - Wrong PIN	Authorization failed, cardholder entered wrong PIN.
809	Authorization declined - Card blocked	Authorization failed, card blocked.
810	Authorization declined - Security issue	Authorization failed, security issue.
811	Authorization declined - Usage control	Authorization failed, usage control.
812	Authorization declined - Double transaction	Authorization failed, current transaction may be a double transaction.
813	Authorization declined - Generic, first AC	Authorization failed, first AC generation failed.
814	Authorization declined - Generic, second AC	Authorization failed, second AC generation failed.
815	Commit timeout	Commit request not sent in time.
816	Rollback not possible	Rollback not possible for current transaction.
817	Cashback amount too low	The intended cashback amount is too low.
818	Cashback amount to high	The intended cashback amount is too high.

Table 118: TIM API related results - Transaction

4 Implementation

4.1 Using .Net

tbd

4.2 Using Java

The TerminalSettings and Terminal classes are in the six.timapi package.

A simple implementation using synchronous calls looks like this:

```
  import com.six.timapi.Terminal;
  import com.six.timapi.TerminalSettings;
  import com.six.timapi.Amount;
  import com.six.timapi.constants.TransactionType;
  import com.six.timapi.constants.Currency;

  class ECRSample {
    public static void main(String [] args) {
      // Create terminal settings to connect to the terminal. Because the empty TerminalSettings() constructor is used, the
global ini-section in the TimApi.cfg file is used.
      TerminalSettings trmSettings = new TerminalSettings();

      //If a specific configuration shall be used for the terminal that will be created, the
TerminalSettings(string:"MyDeviceId") constructor with //DeviceId// parameter can be used.
      TerminalSettings trmSettings = new TerminalSettings("MyDeviceId");

      //After creating the TerminalSettings object it can be adjusted.
      trmSettings.setTerminalId("21001234");
      trmSettings.setConnectionMode(ConnectionMode.OnFixIP);
      trmSettings.setConnectionIPString("127.0.0.1");
      trmSettings.setConnectionIPPort(33335);
      trmSettings.setAutoLogin(true);
      trmSettings.setAutoActivate(true);
      trmSettings.setAutoDeactivate(true);
      trmSettings.setAutoCommit(true);

      // Create terminal instance using the communication settings. These settings can
      // not be changed anymore at a later time.
      Terminal terminal = new Terminal(trmSettings);

      // Set properties affecting the next login and transaction process. Changing POS-ID
      // has no effect until the next logout-login. Changing User-ID affects the next
      // transaction initiated
      terminal.setPosId("POS1234");
      terminal.setUserId("12345678");

      // Define transaction response if required.
      TransactionResponse trxResponse;
      BalanceResponse balanceResponse;

      // Do a purchase transaction
      try {

      // You can call the transaction method directly and if you are not logged in a login() and activate() will be
performed automatically.
      // Because AutoCommit is enabled a commit() is performed during the transaction() method and after the commit() has
been performed the transaction response is returned.
        trxResponse = terminal.transaction(TransactionType.PURCHASE, new Amount(12.50, Currency.CHF));
        // You can also obtain the currency using the language code useful for config files:
        //   Currency.withCode("CHF")

      // If you call the balance method a deactivate() is performed automatically if AutoDeactivate option is enabled.
        balanceResponse = terminal.balance();

      } catch(TimException e) {
        // Request failed. Use e.getRequestType() to learn which request failed.
        // e.getErrorMessage() contains the error message to display
        System.println(e.getErrorMessage());
      }
    }
```

```
  }
```

The same example as above but using asynchonous calls can look as follows:

```
  import com.six.timapi.Terminal;
  import com.six.timapi.TerminalSettings;
  import com.six.timapi.Amount;
  import com.six.timapi.constants.TransactionType;
  import com.six.timapi.constants.Currency;

  import javax.swing.*;

  class ECRSample {
    public static void main(String [] args) {
      // Create terminal settings as described in the example above!
      TerminalSettings trmSettings = new TerminalSettings();

      // Create terminal instance using the communication settings. These settings can
      // not be changed anymore at a later time.
      Terminal terminal = new Terminal(trmSettings);

      // Set properties affecting the next login and transaction process. Changing POS-ID
      // has no effect until the next logout-login. Changing User-ID affects the next
      // transaction initiated
      terminal.setPosId("POS1234");
      terminal.setUserId("12345678");

      //Add the notifiers you wish to handle. It can be done by adding a subclass of six.timap.TerminalNotifier as follows:

      // Add activateCompleted, transactionCompleted, commitCompleted, deactivateCompleted and balanceCompleted notifier if
required.
      terminal.addNotifier(new TerminalNotifier() {
       @Override
       public void activateCompleted(TimEvent e, ActivateResponse data) {
       // my code to be called if a activate request completed.

       }

       @Override
       public void transactionCompleted(TimEvent e, TransactionResponse data) {
       // my code to be called if a transaction request completed.

       // After the transaction request is completed correctly, a commit can be performed. Only required if AutoCommit is
not enabled.
       SwingUtilities.invokeLater(new Runnable() {
         public void run() {
           terminal.commitAsync();
         }
       });
       }

       @Override
       public void commitCompleted(TimEvent e) {
       // my code to be called if a commit request completed.

       // A deactivateAsync() can be called here. This is not required if the option AutoDeactivate is enabled, it will then
be performed automatically.
       SwingUtilities.invokeLater(new Runnable() {
         public void run() {
           terminal.deactivateAsync()
         }
       });

       // If AutoDeactivate is enabled, a balanceAsync() can be called directly and a deactivation will be performed
automatically.
       SwingUtilities.invokeLater(new Runnable() {
         public void run() {
           terminal.balanceAsync()
         }
       });
       }

       @Override
       public void deactivateCompleted(TimEvent e, DeactivateResponse data) {
       // my code to be called if a deactivate request completed.
       }

       @Override
       public void balanceCompleted(TimEvent e, BalanceResponse data) {
       // my code to be called if a balance request completed.
       }
      });
```

```
      // Start a transaction process
      // If the terminal is not yet logged in or activated this two actions are performed automatically before performing
the transaction.
        terminal.transactionAsync(TransactionType.PURCHASE, new Amount(12.50, Currency.CHF));

        // If the option AutoCommit is enabled the commitAsync() method is called automatically after the transaction has
finished.
        // If not enabled, the commitAsync() method shall be called after receiving the transactionCompleted event.

        // If the commit request is completed correctly the deactivate can be started (see commitCompleted method).
        // If the AutoDeactivate option is enabled a deactivateAsync() is not required to be called.
        // Another request that need a Deactivated state will then perform a deactivateAsync() automatically. Otherwise the
deactivateAsync() method can be called after a successful commitCompleted event.

        //So if AutoDeactivate is enabled, a balanceAsync() can be called directly. This can be done e.g. in the
commitCompleted event (see commitCompleted event).
    }
  }
```

Attention: *Events are send from inside a thread different than the main thread. If you need to access code in the event handling thread use something like java.awt.EventQueue.invokeLater or javax.swing.SwingUtilities.invokeLater*

4.3 Using Swift

The Swift binary API is in development and will be described in a later document Version.

5 Appendix

5.1 Data Types

The following table contains the basic data types.

Abbreviation	Description
int	Integer value.
String	String using explicit length. Unless stated otherwise any value including the 0-character is allowed.
boolean	Boolean value.
Date	Object representing a date. Supports year, month and day indication.
DateTime	Object representing a timestamp. Combination of Date and Time data type.
Time	Object representing a time. Supports hour, minute and second indication.
void	Empty data type used for functions returning no value. Depending on the language this can be a None object.
List<*DataType***>**	An ordered list of instances of type *DataType*. Language specific implementation.
Map<*DataTypeKey***,** *DataTypeValue***>**	An mapping between instances of type *DataTypeValue* keyed by instances of type *DataTypeKey*. *DataTypeKey* has to be immutable. Language specific implementation.
EnumSet<*Enumeration***>**	A list of enabled enumeration constants. Used for enumerations representing flags that can be individually enabled or disabled. Language specific implementation

Table 119: Data Types

5.2 Configuration File

With the use of the configuration file “TimApi.cfg” GlobalSettings and CommunicationSettings can be set from outside of the application. If the user wishes to use multiple instances of the *Terminal* class it can be configured using ini-sections in the config file which are separated with a user defined *DeviceId*. The same *DeviceId* has to be used in the constructor of the *TerminalSettings* class if the specified configuration is wished to use. If no *DeviceId* is specified the global ini-section is used.

The precedence of the configuration is as follows:

1. TimApi.cfg will be read first by creating a *TerminalSettings* object
2. The config previously read from the TimApi.cfg file can be overritten by hand using setter methods from the *TerminalSettings* class.

Example “TimApi.cfg”: This example described the TimApi.cfg file with a global ini-section and two different ini-sections for a specific *DeviceId*.

```
[global]
ProtocolType SIXMLTCP
TerminalId 21001234
LogDir var/log
AutoConnect On

[MyDeviceId]
ProtocolType SIXMLTCP
TerminalId 12345678
LogDir var/otherLog
AutoConnect Off

[MySecondDeviceId]
ProtocolType SIXMLTCP
TerminalId 87654321
LogDir var/yetAnotherLog
ConnectionMode OnFixIp
```

Parameter	Description
LogDir	Directory where log files are generated
TerminalId	Terminal ID to be broadcasted in case of ConnectionMode Broadcast
ConnectionMode	Broadcast (default) or OnFixIP
ConnectionIPString	IP address of the EFT terminal in case of ConnectionMode OnFixIP.
ConnectionIPPort	Listening Port of the EFT terminal in case of ConnectionMode OnFixIP
ProtocolType	Protocol type
AutoConnect	Automatically connects after a new instance of Terminal API is created or after the connection to the EFT has been lost. Does not automatically connect if the API user calls Disconnect() manually. Default is on (value “On” or “Off”)
AutoLogin	Automatically logs in after connected to the EFT. Default is on (value “On” or “Off”)
FetchBrands	Automatically retrieves application information during logging in. Default is on (value “On” or “Off”)
AutoActivate	Automatically activate EFT after every request finishes that is applicable in logged in state. This affects any situation such a request is issued. Also automatically activating after logging in. Does not affect calling Deactivate() manually. Default is on (value “On” or “Off”)
AutoCommit	After executing the Transaction- function the API commits the transaction automatically. Default is on (value “On” or “Off”)
AutoDeactivate	Automatically deactivate EFT after every request finishes that is applicable in open state. This affects any situation such a request is issued. Does not affect calling {@link #deactivate} manually. Default is on (value “On” or “Off”)
AutoLogout	Automatically logs out after Deactivate() has been called or an Auto-Deactivate has been performed. Default is on (value “On” or “Off”)
AutoDisconnect	Automatically disconnect after Logout() has been called or an Auto-Logout has been performed. Default is on (value “On” or “Off”)

Table 120: Configuration File Parameters