gematik

# OpenHealthCard

## *UserGuide*

gematik GmbH

Version 0.4.0-SNAPSHOT, 2020-01-22 08:13:20 +0100 Uhr: Working Document

# Table of Contents

# Preface

## Objective

This document describes the interface of the German Healthcard System library for mobile apps (OpenHealthCard API).

## Target Group

The document is targeted to software developers using this library to create mobile apps in order to create applications based on the German Healthcard and the underlying Telematics Infrastructure to be used by health professionals.

## Scope

The document is limited to the library and intends to describe the library from the developer's point of view.

## Assignment

This document does not contain any regulations on the German Healthcard System.

## Methodology and Conventions

The document starts with a description of the architecture of the library :globalConfig: true :toc-title: Table of Contents

# Chapter 1. Introduction

## 1.1. Purpose

This API should ease development of mobile Apps to provide following applications of the German Healthcard System by implementation of core functionality described in the specification bundle published by Gematik:

- personal insurant master data
- emergency data
- medication data

It provides a standardized API to access these data stored on the German Healthcard.

German Healthcard System bases on mainly three different smartcard types to be used in conjunction:

- health insurance card (healthcard, eGK))
- health professional card (HPC, HBA)
- secure module card type B (SMC-B)

This API provides interfaces to access those cards when connected to card reader in order to read data from healthcard or to access smartcard functionality for cryptographic operations, like authentication or encryption.

## 1.2. System Context

Following picture depicts the System Context describing dependencies to the outside world of this library.

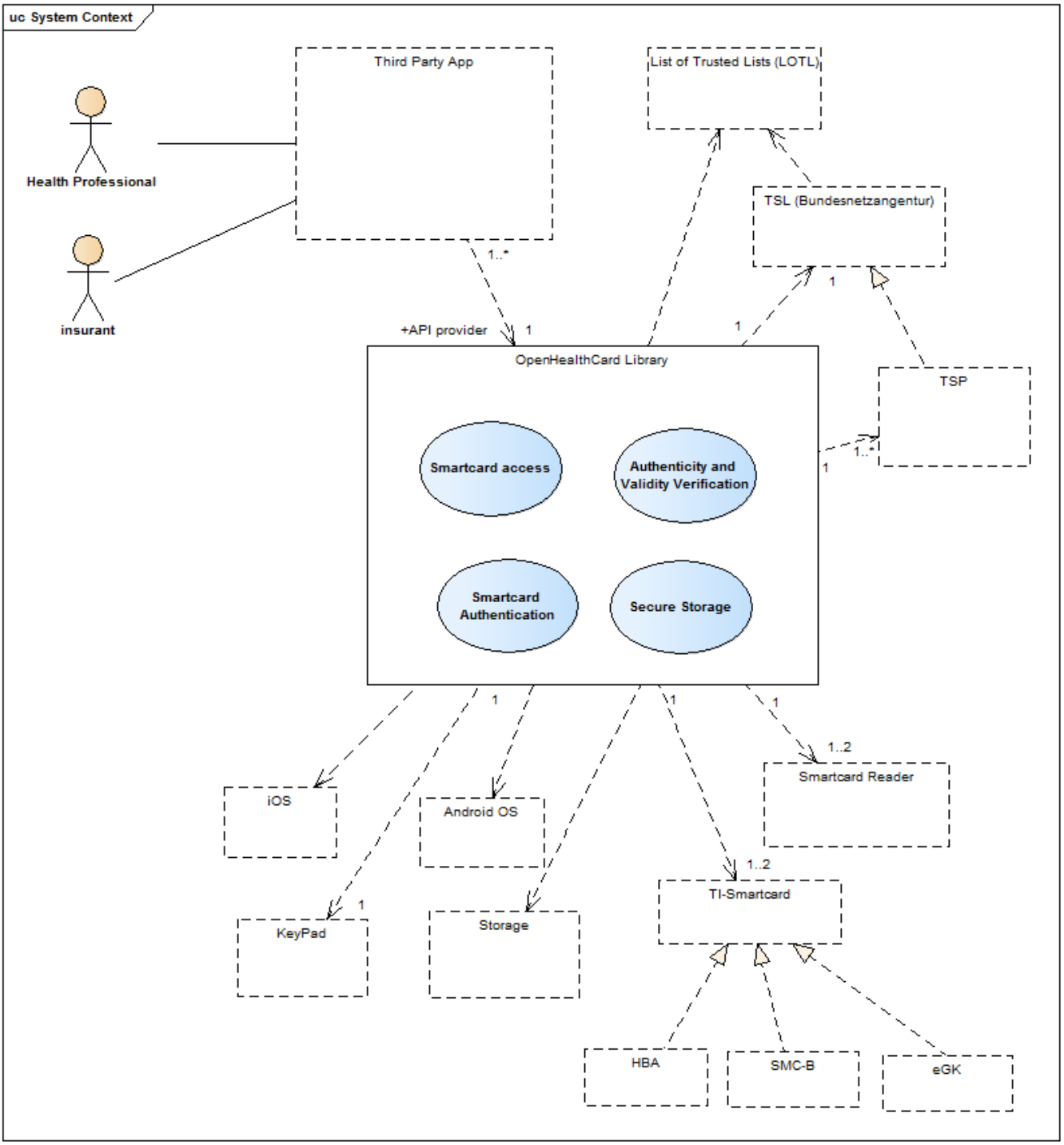



*Figure 1: System-Context*

There are following main groups of use cases the library supports:

- Smartcard access
- Authenticity and Validity Verification
- Smartcard Authentication
- Secure Storage

The library makes use of Android or iOS operating system functionality in order to access smartcards inserted into card readers which in turn are connected to the mobile device. Card readers can be connected to the device via

- USB
- Bluetooth

Smartcards can also be connected to the device via builtin NFC support.

### 1.2.1. Smartcard access

The library supports developers by sending commands to one of the smartcard types in order to read and write data from or to smartcard.

It also detects the type and version of the smartcard connected to.

Applications can register at the library to be get notified on a smartcard or card readers connected to the mobile device.

### 1.2.2. Authenticity and Validity Verification

The library provides functionality to check the authenticity and validity of a smartcard. This can be done by verification of card holder's certificate or by mutual authentication between two smartcards connected to the mobile device at same time.

### 1.2.3. Smartcard authentication

This functionality eases the usage of the smartcard for authentication purposes, e.g. for authentication on web portals.

### 1.2.4. Secure storage

In order to store data previously read from a secure storage like a smartcard on a mobile device, those data need to be securely encrypted and decrypted. This is what this functionality provides in an easy-to-sue way based on the cryptographic functionality of smartcards.

## 1.3. Architectural Layers

The architecture of the library is divided into three main layers.

- CardReaderProvider API
- HealthCard Access API
- HealthCardControl API

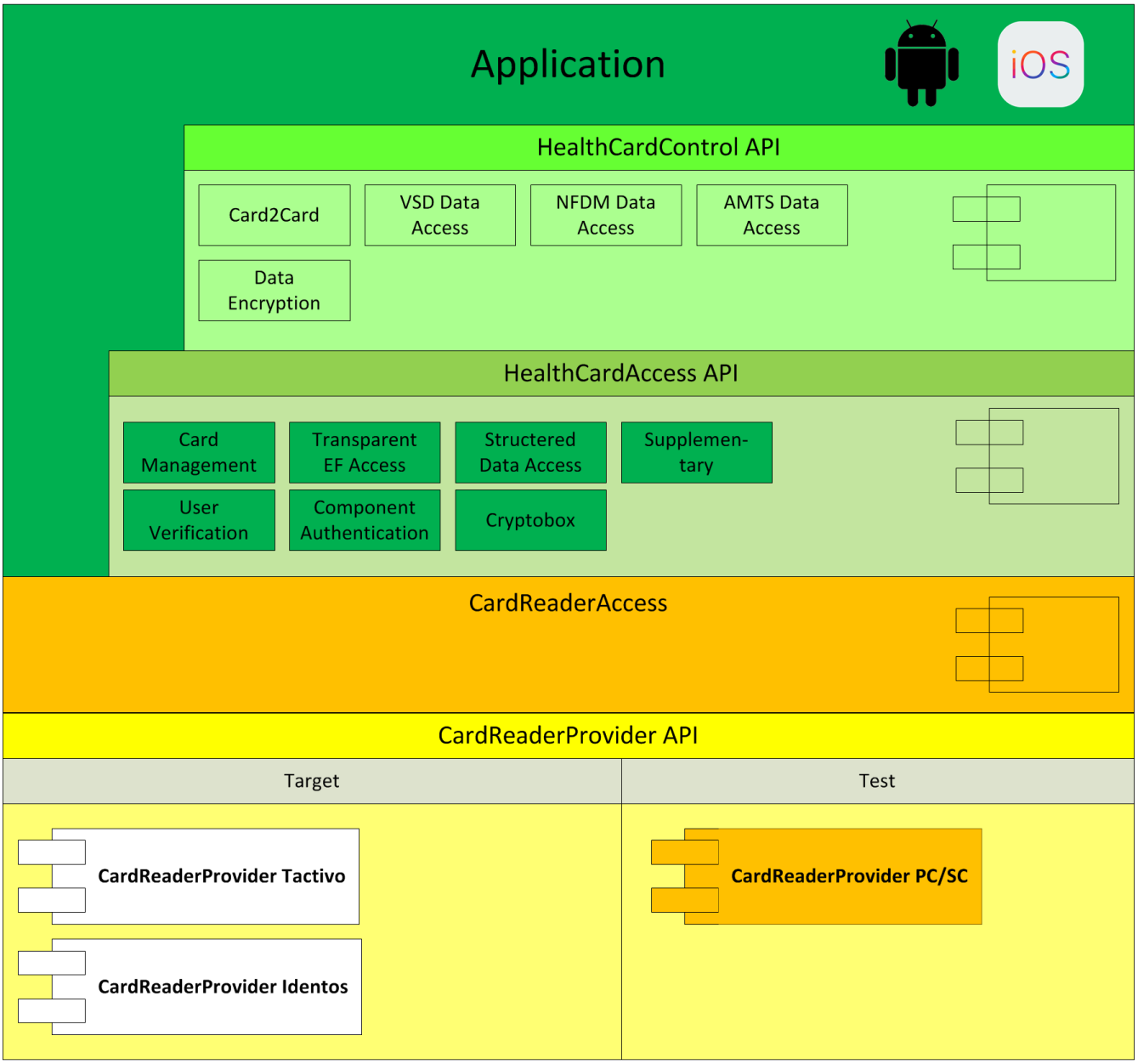



*Figure 2: Architectural Layers*

### 1.3.1. HealthCardControl API

This is the main API on the highest level of abstraction of physical access providing functionality to gain access to application data on a healthcard including authentication and verification.

### 1.3.2. HealthCardAccess API

This layer provides low level access to smartcard interfaces by in order to send a command to the smartcard. All available smartcard commands for those smartcards of German Healthcard system are defined and can be configured to be sent. This layer generates APDU byte arrays from command and command parameters.

Smartcard commands can be chained. There can also expected responses values included into the change prior to the next command. Thus command chains can be broken if an unexpected response was received from a smartcard as a result of a command.

### 1.3.3. CardReaderProvider API

This layer provides the possibility to attach a smartcard reader to the HealthCardAccess API layer. For each card reader, i.e. each card reader SDK, there is an adapter necessary to be implemented in order to provide the CardReaderProvider interface used to attach this smartcard reader to the OpenHealthCard library.

The library comes with a set of card reader adapters supporting following card readers:

- Identos ID1xx
- Precise Biometrics Tactivo

## 1.4. General Approach

# Chapter 2. HealthCardControl

This part describes the usage of HealthCardControl.

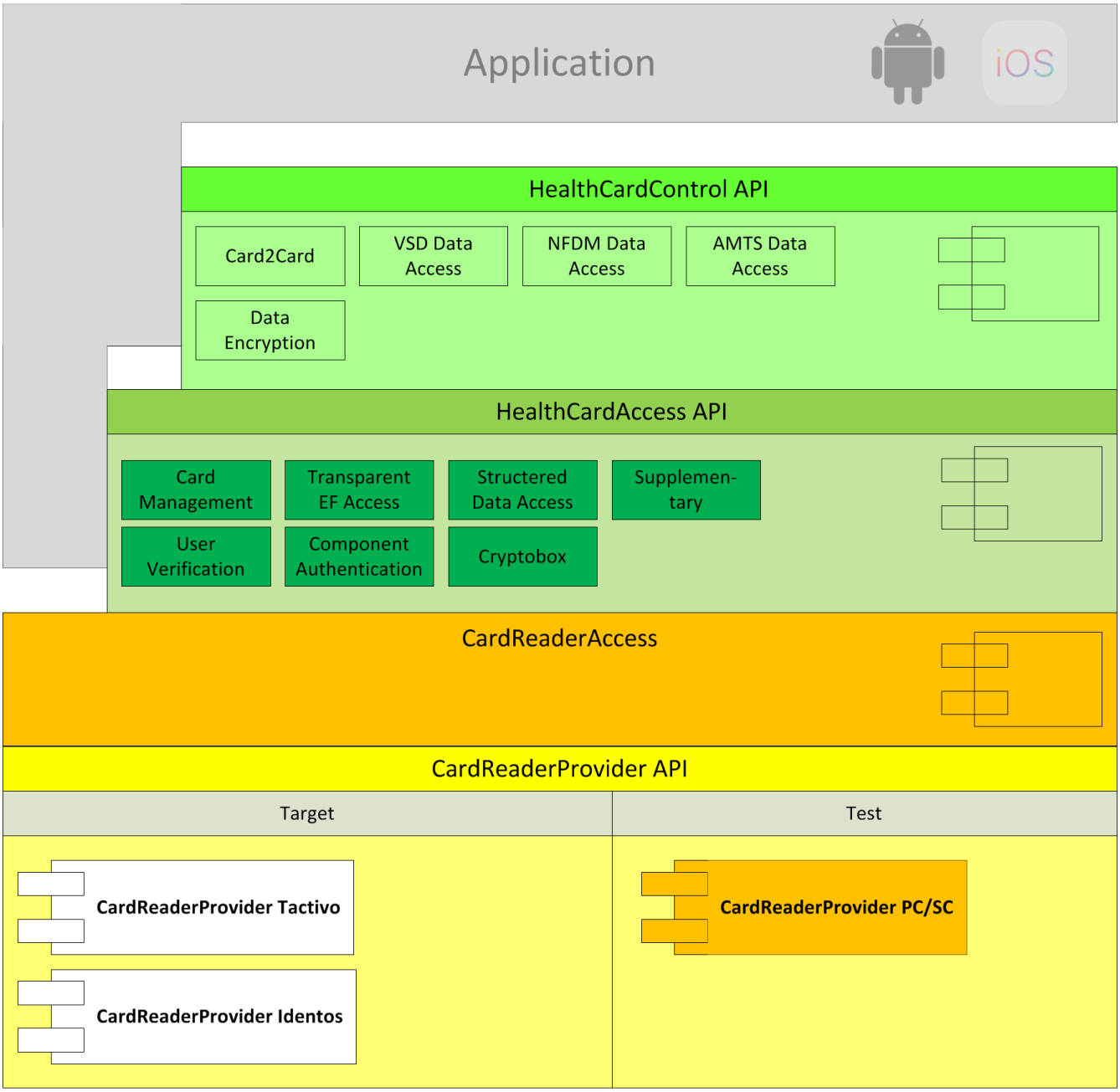
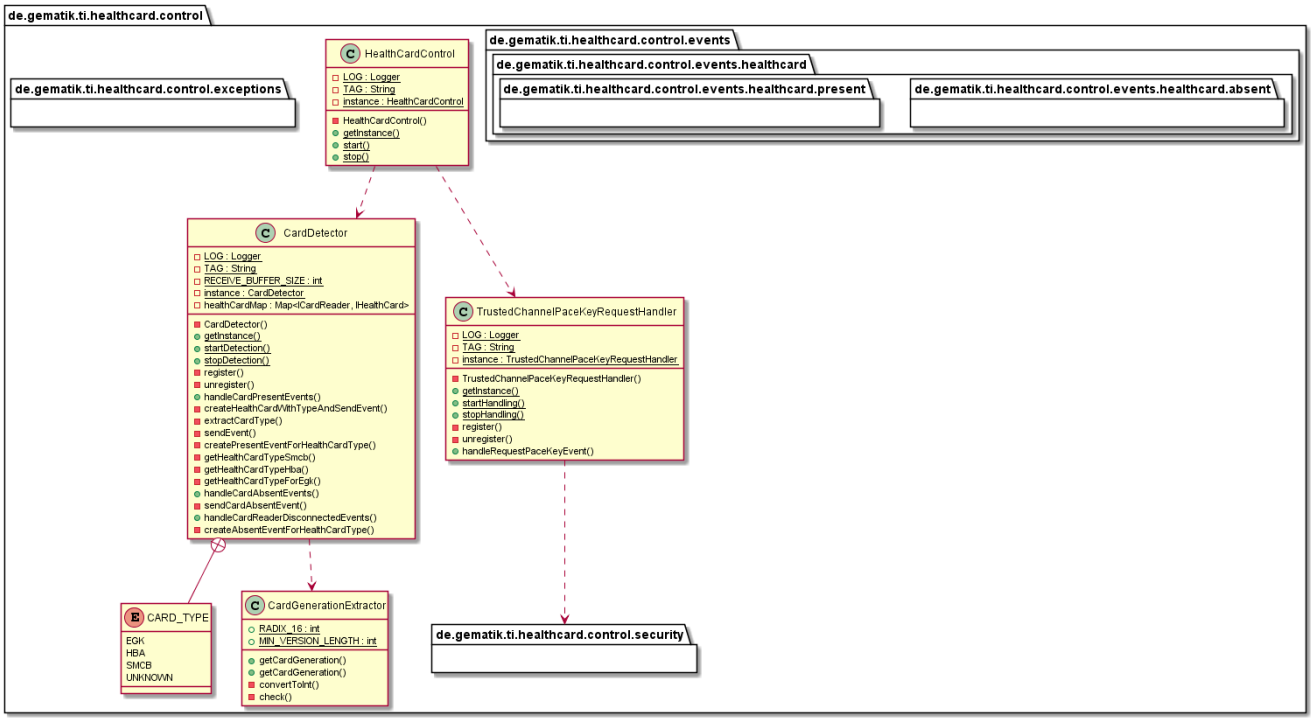



*Figure 3: HealthCardControl layer API*

## 2.1. Structure

### 2.1.1. Overview

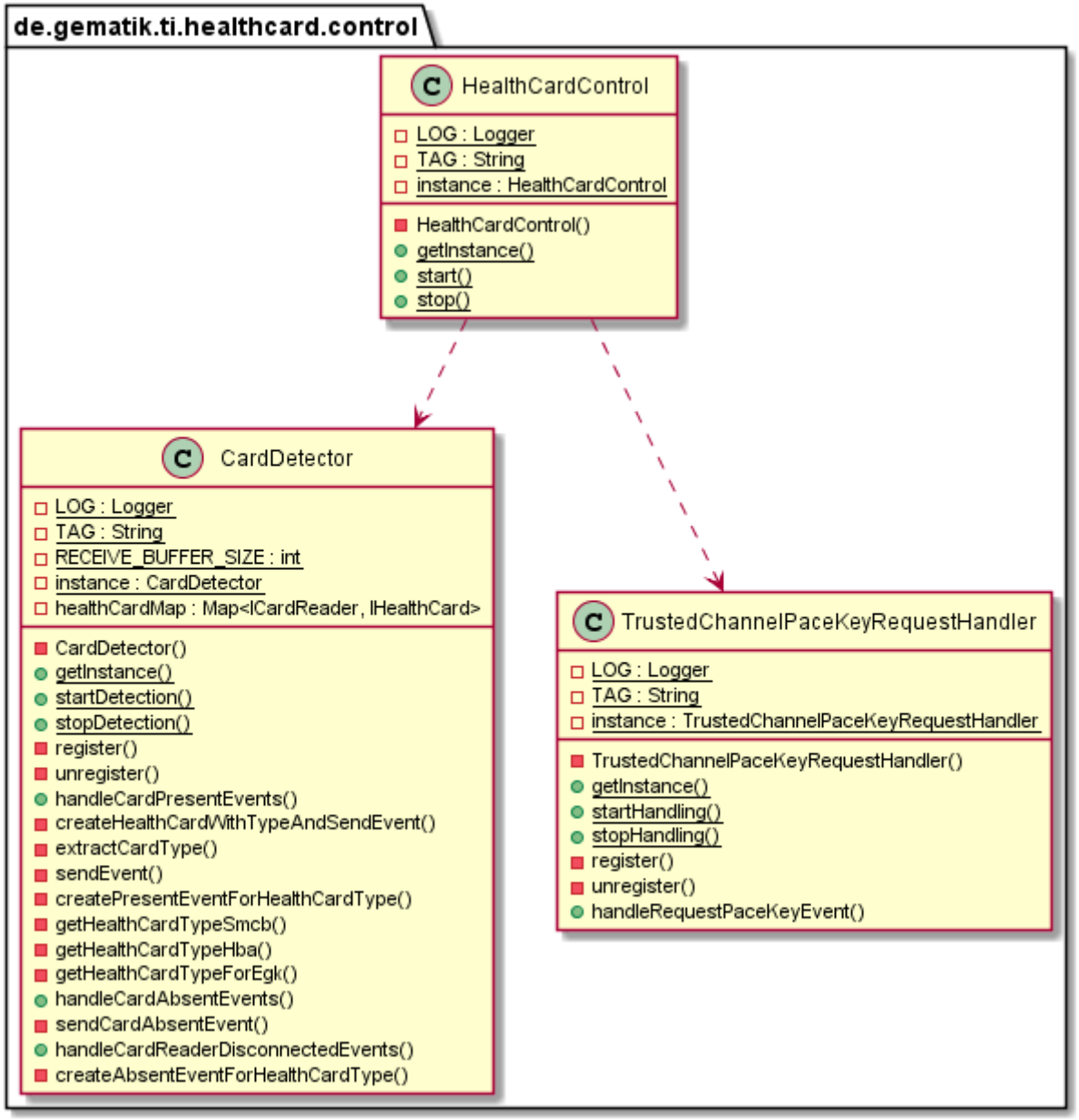
This library contains the `CardDetector` to determine automatically the current present card type and generation and inform about card events. The Events send over EventBus are stored in package `de.gematik.ti.healthcard.control.events` and subpackages.




*Class diagram 1: Overview of health card control*

## 2.1.2. HealthCardControl

This is the singleton object to start all handler and services for the control layer from one point

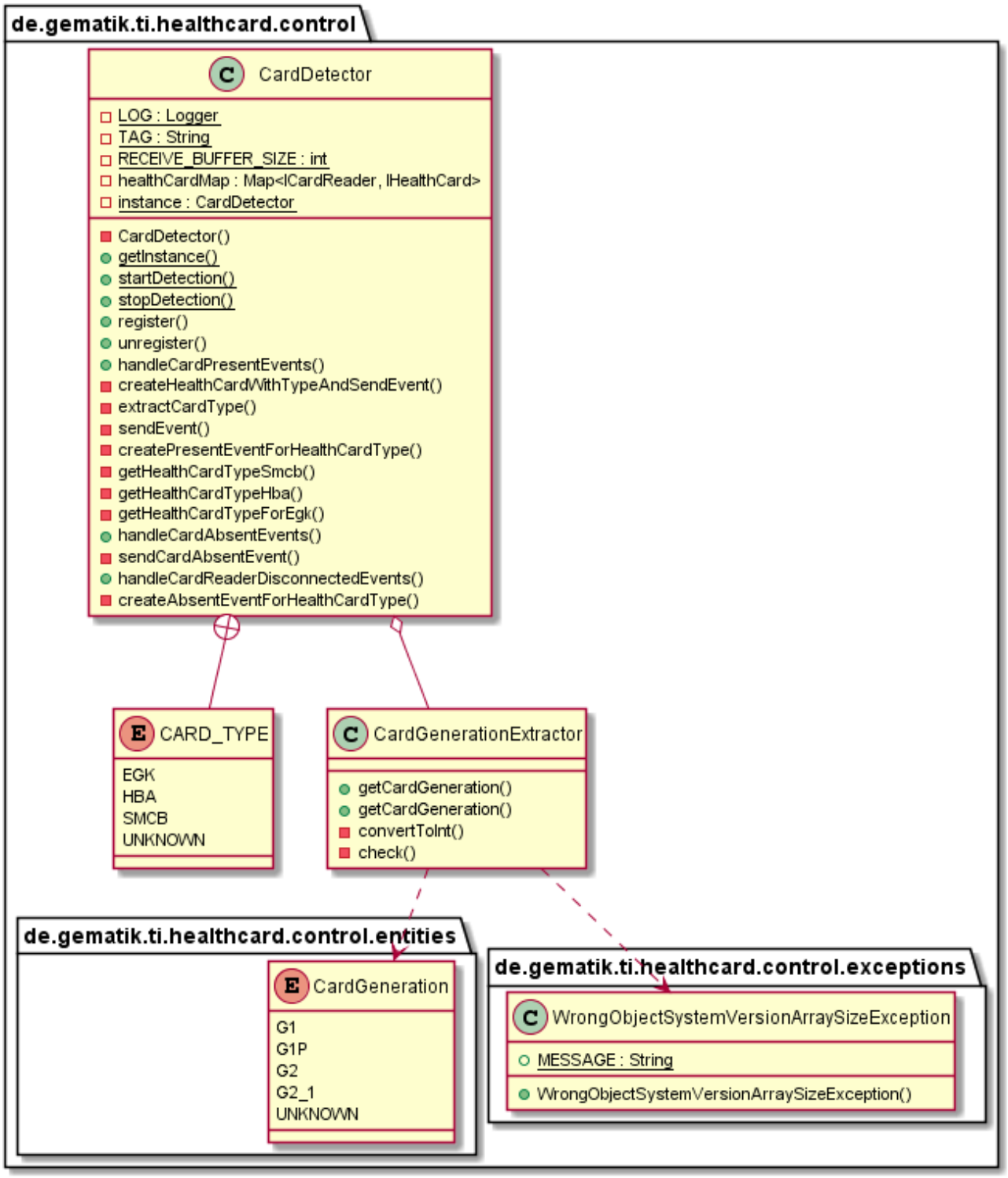



*Class diagram 2: HealthCardControl*

## 2.1.3. CardDetector

### 2.1.3.1. Purpose

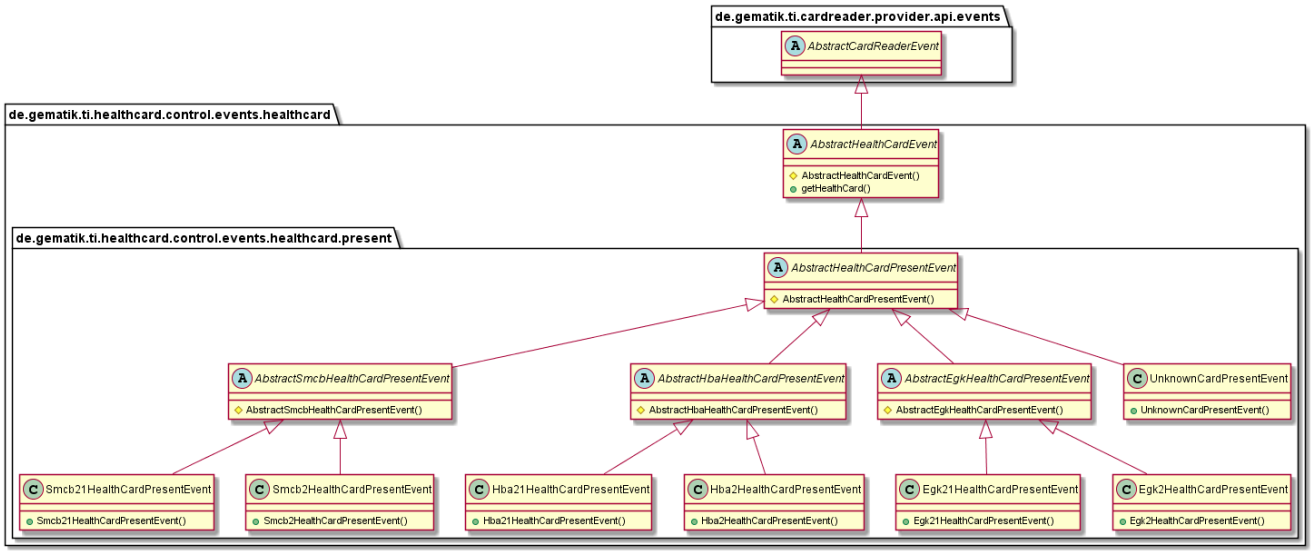
The singleton Card Detector subscribe to EventBus for Card- and CardReader-Events. For each CardPresent-Event determine this class automatically the Card Type (e.g. EGK, HBA or SMCB) and the Card Generation (e.g. G2, G2.1). With this information would the `HealtCard` object initialized and the EventBus subscriber informed with HealthCardPresentEvents for the specific type. Furthermore the CardDetector send an event for each absent card or disconnected card reader with containing health card.




*Class diagram 3: CardDetector*

#### 2.1.3.2. HealthCardPresentEvents

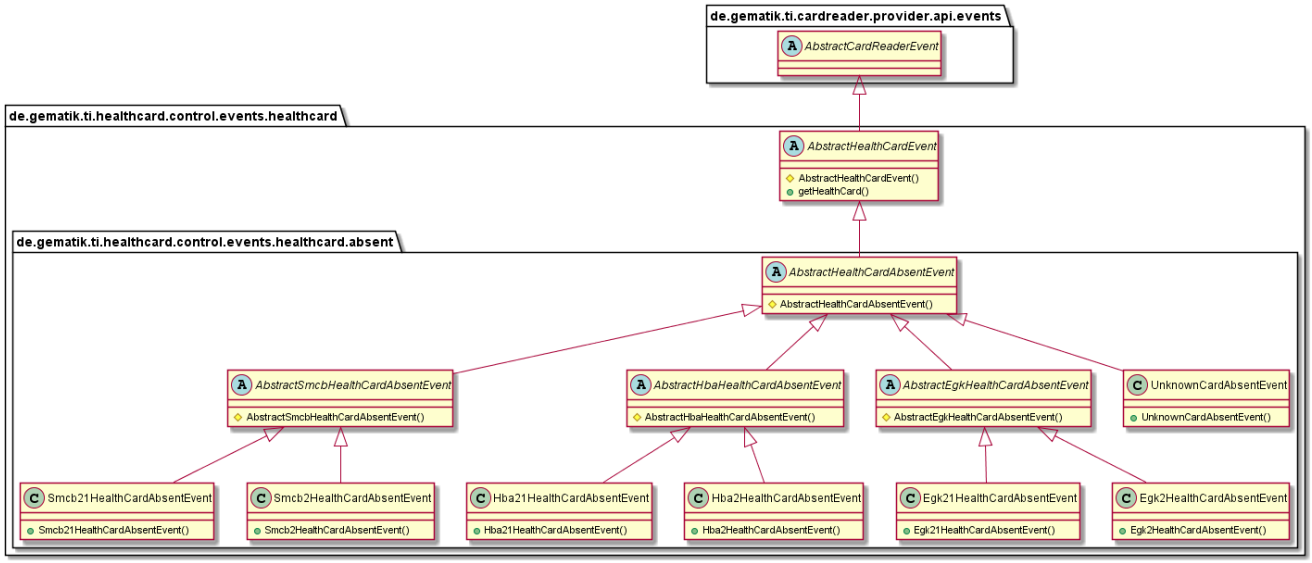
The control layer send specific events for each present health card or if the type is unknown an UnknownCardPresentEvent. The subscriber could subscribe for specific HealthCardPresentEvent e.g. Egk21HealthCardPresentEvent or for generally events like AbstractEgkHealthCardPresentEvent for all EGK present events.




*Class diagram 4: HealthCardPresentEvents*

#### 2.1.3.3. HealthCardAbsentEvents

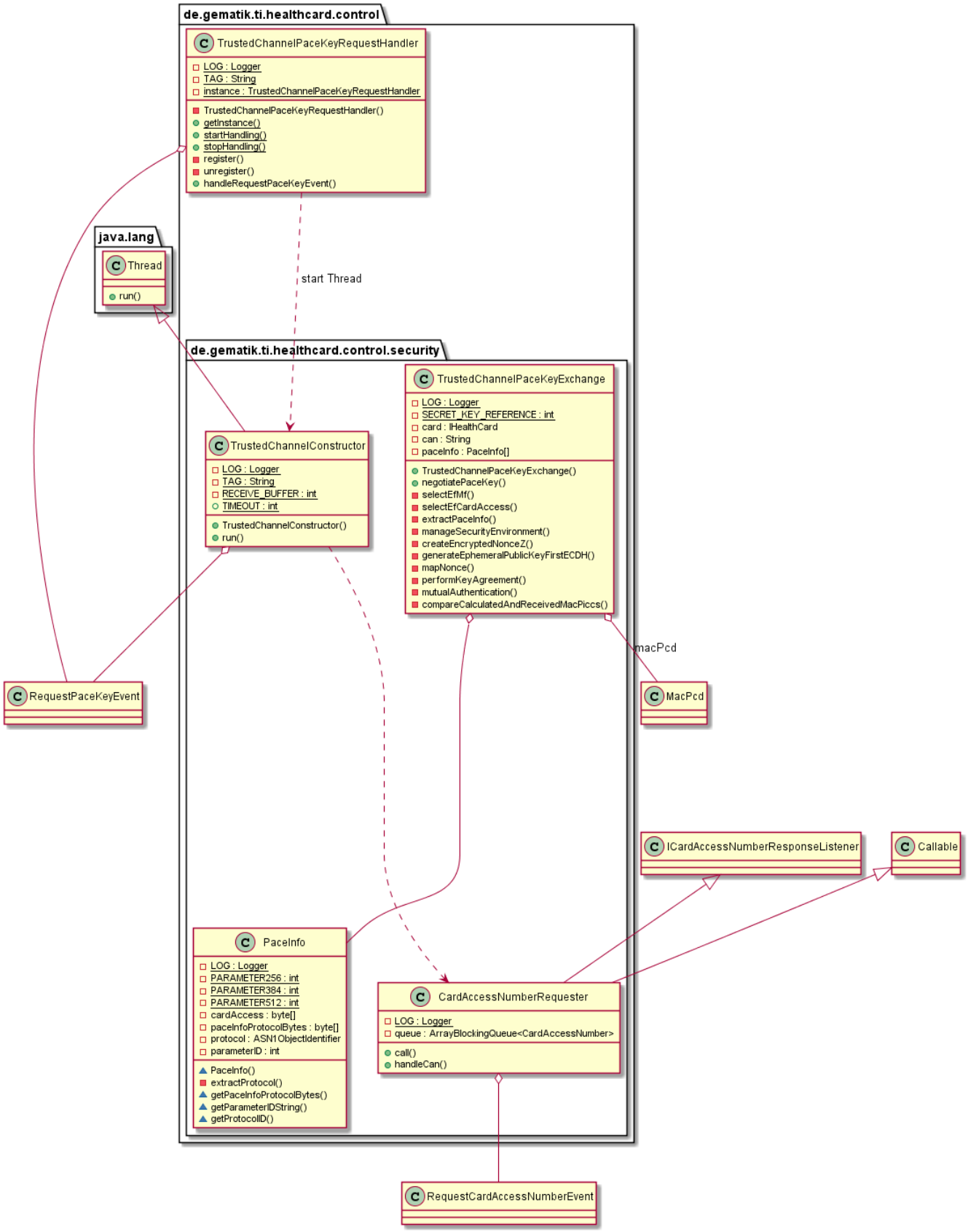
The control layer send specific events for each absent health card or for disconnected card reader with containing health card. The subscriber could subscribe for specific HealthCardAbsentEvent e.g. Egk21HealthCardAbsentEvent or for generally events like AbstractEgkHealthCardAbsentEvent for all EGK absent events.




*Class diagram 5: HealthCardAbsentEvents*

### 2.1.4. TrustedChannelPaceKeyRequestHandler

The singleton TrustedChannelPaceKeyRequestHandler subscribe to EventBus for PaceKey-Request-Events. For each PaceKeyRequest-Event start this instance a thread to request the CardAccessNumber over Event-Bus request from UI or other application. After CardAccessNumber response starts the PaceKey negotiation and after success negotiation would the requester informed about the pacekey.




*Class diagram 6: TrustedChannelPaceKeyRequestHandler*

# 2.2. Getting Started

### 2.2.1. Build setup

To use CardReaderControl library in a project, you need just to include following dependency:

*Listing 1. Gradle dependency settings to use CardReaderControl library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'healthcard.control', version:
'1.0.0-SNAPSHOT'
}
```

*Listing 2. Maven dependency settings to use CardReaderControl library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>healthcard.control</artifactId>
        <version>{version_healthcard_control}</version>
    </dependency>
</dependencies>
```

# Chapter 3. HealthCardAccess API

## 3.1. Introduction

This part describes the usage of low level HealthCardAccess API in order to send smartcard commands to a smartcard connected to card reader.

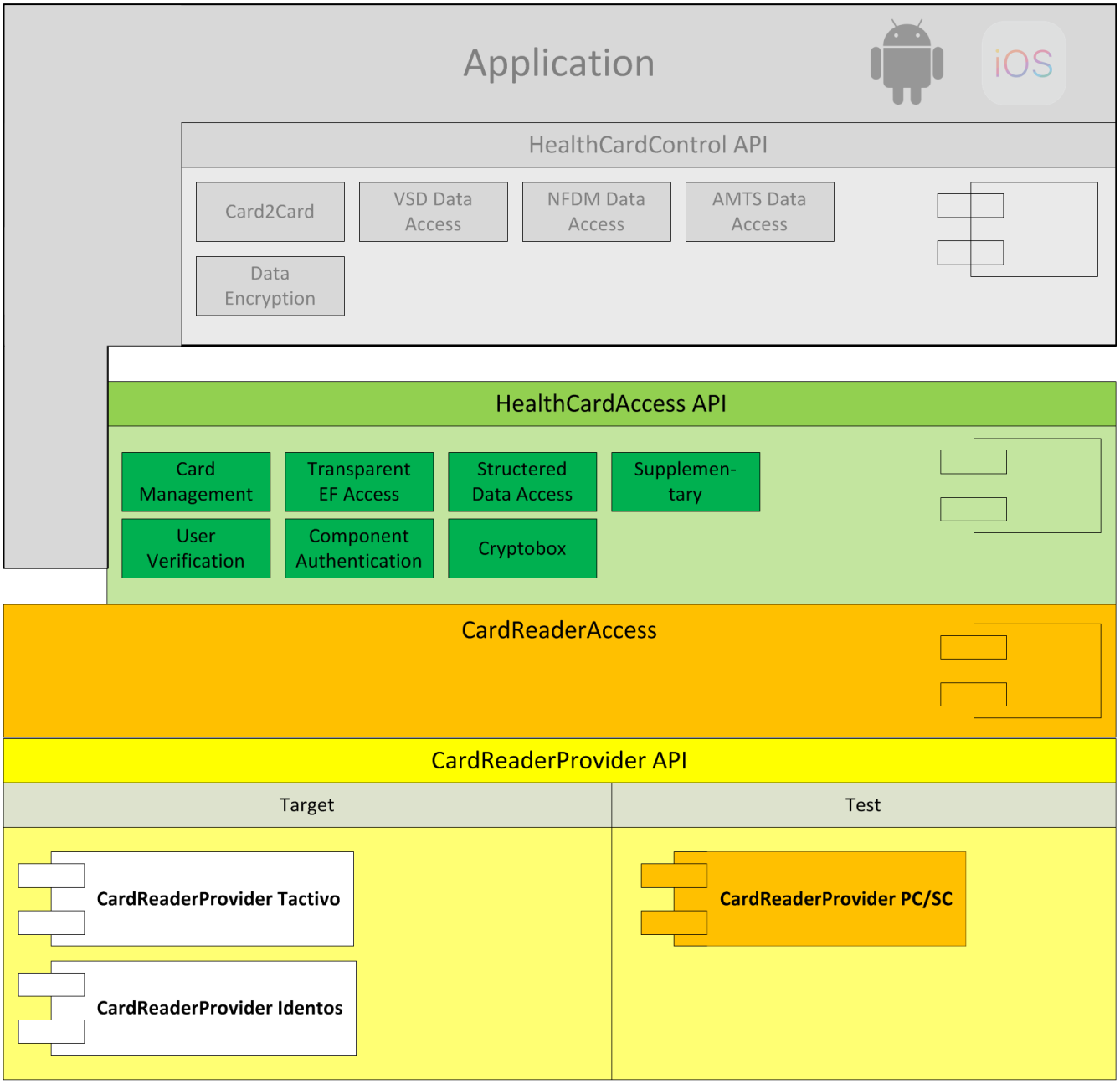
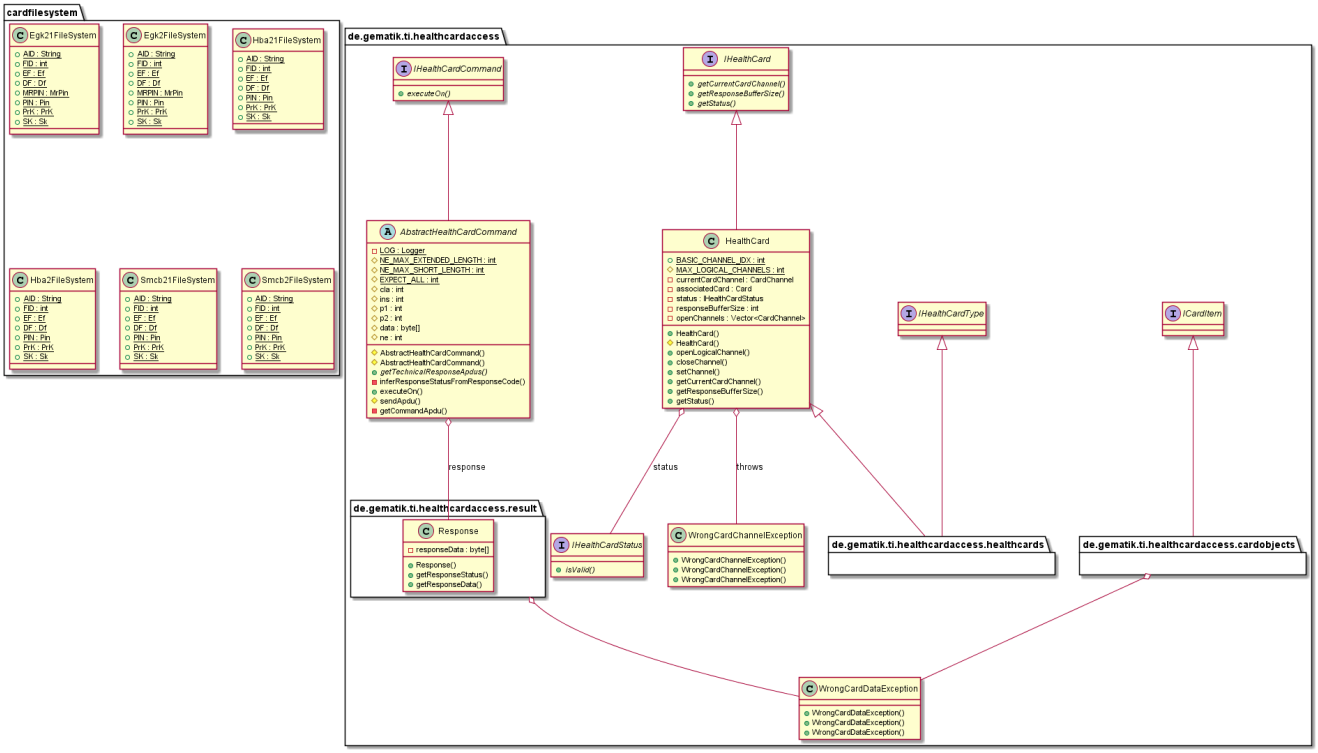



*Figure 4: HealthCardAccess layer API*

## 3.2. Overview

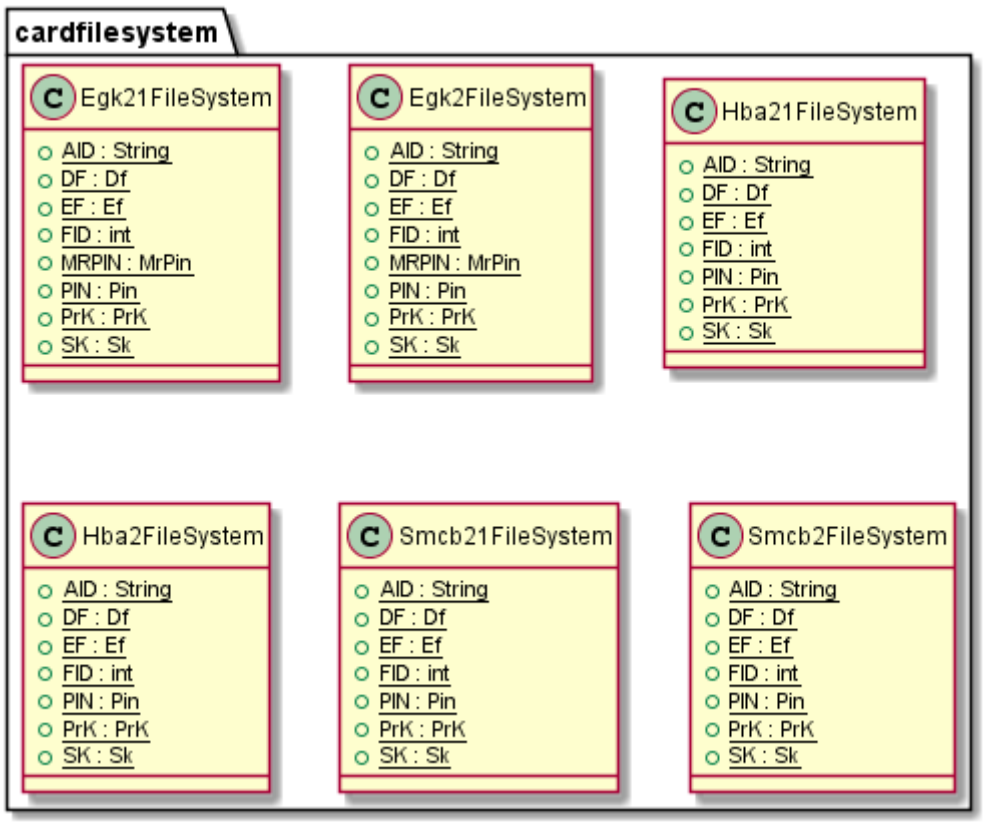
This library contains an package `cardfilesystem` which contains the card structure as classes for each supported health card type. Furthermore the `de.gematik.ti.healtcardaccess` package and subpackages contains the classes for cards, commands and error handling.




*Class diagram 7: HealthCardAccess Overview*

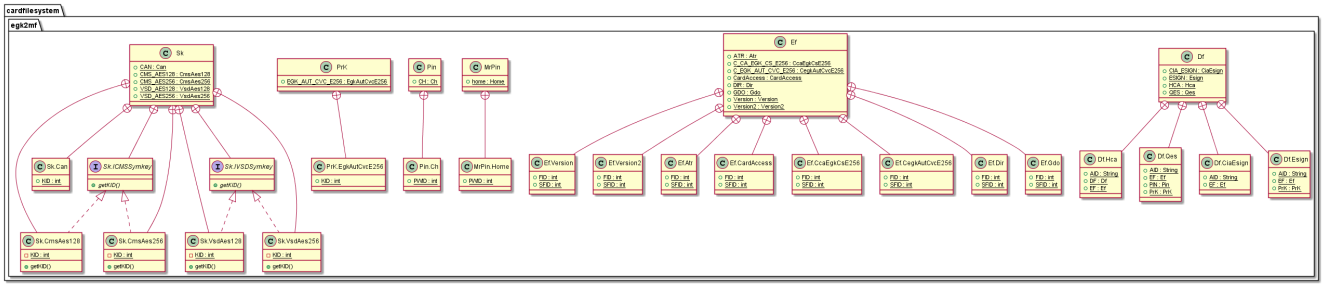
### 3.2.1. CardFileSystem

The CardFileSystem represents the structure of health cards (EGK, HBA, SMCB, SMCK) and for specific generation (G2, G2.1).



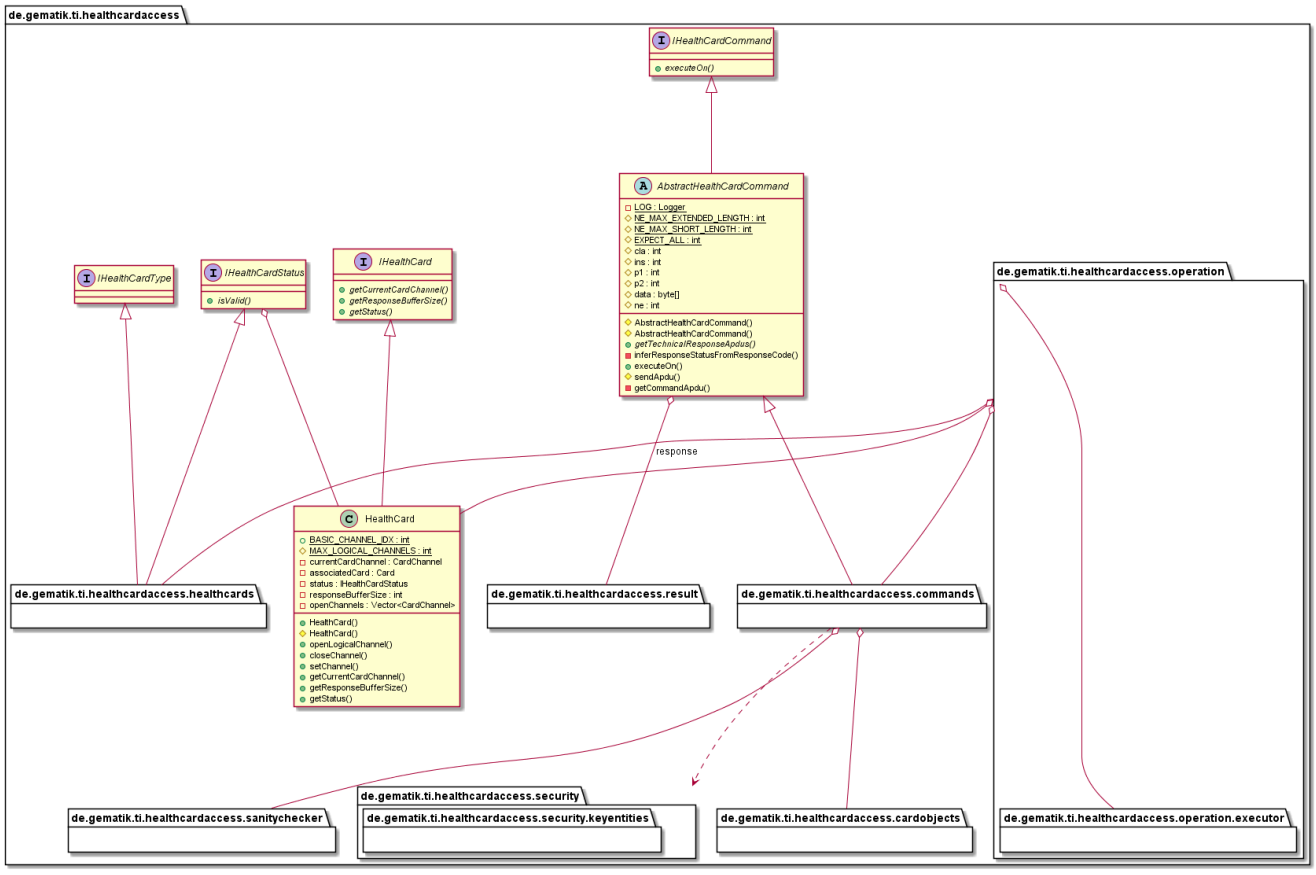

*Class diagram 8: CardFileSystem*

The entry object is one of `{TYPE}{GENERATION}FileSystem`. This contains the root structure for each type of card system. From this root point could you select the child's and so on. The following picture show exemplary for all FileSystems the complete EGK2 FileSystem structure. .EGKG2 FileSystem



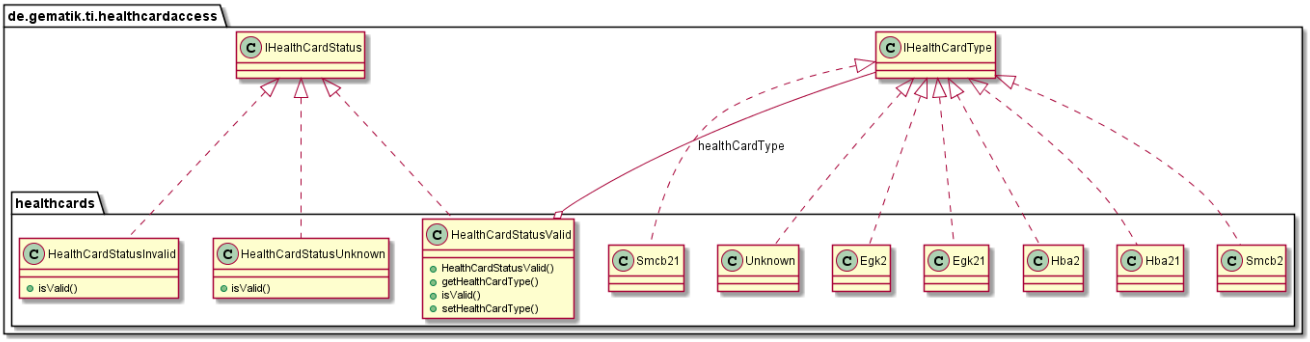

## 3.2.2. HealthCardAccess API

The HealthCardAccess API Structure contains `healthcards` package with object represents all supported card types, `commands` and `result` package with all supported commands and responses for health cards, `cardobjects` package with possible object on health cards and `operation` package with functionalities to cascade and execute commands on health cards. .Packages HealthCardAccess API




### 3.2.2.1. Health Cards

The health card objects in `healthcards` package represent the potential types of health cards and this type is indirect stored on HealthCard objects with `IHealthCardStatus`. The implementation of `IHealthCardStatus` could `HealthCardStatusValid` and this stores the health card type.

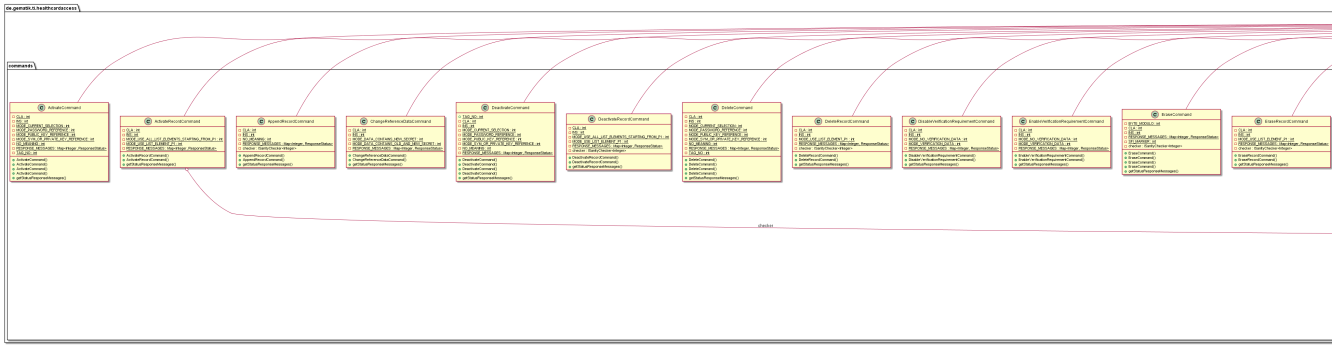



*Class diagram 11: Health Cards*

Furthermore contains the `HealthCard` object the physical card from card reader, response buffer and channels.

#### 3.2.2.2. Commands

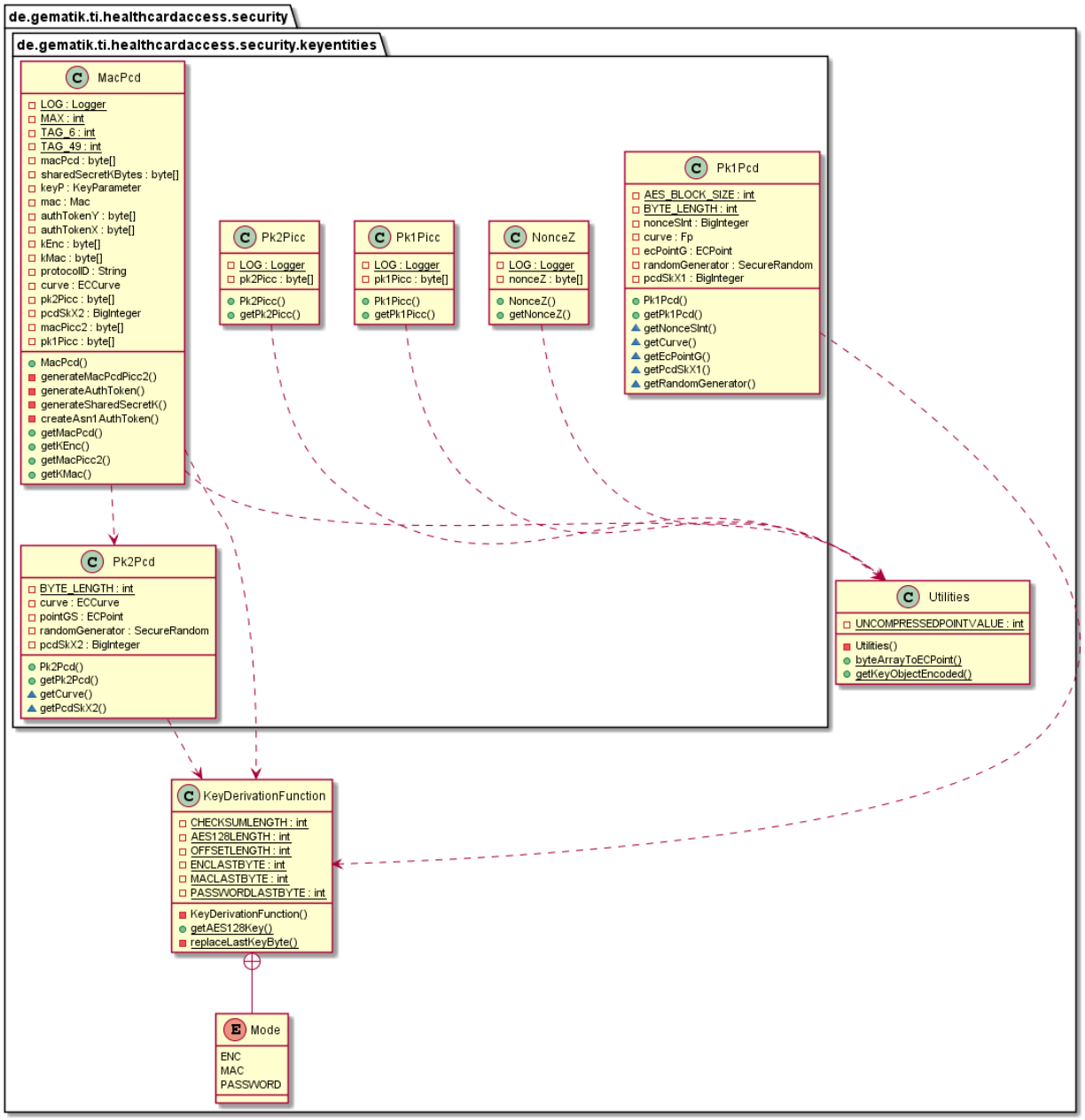
The `commands` package contains all available commands for health cards.



*Class diagram 12: Commands*

#### 3.2.2.3. Security

The security and sub-package contains the key derivation function and the key entities. .Security




# 3.3. Getting Started

## 3.3.1. Create a command

The design of this API follows the command design pattern as well as functional programming paradigm comparable to ReactiveX. Thus a command object of appropriate command class for the desired command to be sent has to be created first. When creating the command it needs to be configured to be configured.

Following example shall send a SELECT command to a smartcard in order to select the folder (aka Distinguished File = DF) DF.HCA. So we need to to create a SelectCommand object an pass the FileIdentifier of DF.HCA to the command object.

```
AbstractHealthCardCommand hcc = new SelectCommand(Egk.Df.HCA.getAid());
```

As commands are executed on smartcards, such command object can be executed on a card object. Next example presumes that we already got an Egk object as one kind of a healthcard. The result of command execution can be validated against an expected response status, e.g. SUCCESS response status. In order to test the response status it needs to be retrieved from the execution response. This is done by mapping `ResultOperation<Response>` to `ResultOperation<ResponseStatus>` and to test it against the expected `ResponseStatus` by means of `TestSubscriber` object created by calling `test()` method . This is shown in next example.

```
private Egk hc;
...

ResultOperation<Response> result = hcc.executeOn(hc);
ResultOperation<Response.ResponseStatus> status =
result.map(Response::getResponseStatus);
TestSubscriber<Response.ResponseStatus> subscriber = status.test();
subscriber.assertValue(Response.ResponseStatus.SUCCESS);
```

If command execution returns a different value than SUCCESS as expected in example above, an `AssertionError` is thrown at the end of command execution.

The example above can be written in a much shorter format by omitting the object references and chaing the method executions, like shown below.

```
private Egk hc;
...

hcc.executeOn(hc)
                .map(Response::getResponseStatus)
                .test()
                .assertValue(Response.ResponseStatus.SUCCESS);
```

### 3.3.2. Build setup

To use HealthCardAccess library in a project, you need just to include following dependency:

*Listing 3. Gradle dependency settings to use healthcard access library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'healthcard.access', version: '1.2.1'
}
```

*Listing 4. Maven dependency settings to use healthcard access library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>healthcard.acces</artifactId>
        <version>1.2.1</version>
    </dependency>
</dependencies>
```

# Chapter 4. CardReaderAccess API

## 4.1. Introduction

This part describes the usage of CardReaderAccess API in order to use different card reader and provider in your application. The CardReaderAccess API provides a registry for higher layer applications to inform about new connected and disconnected card reader.

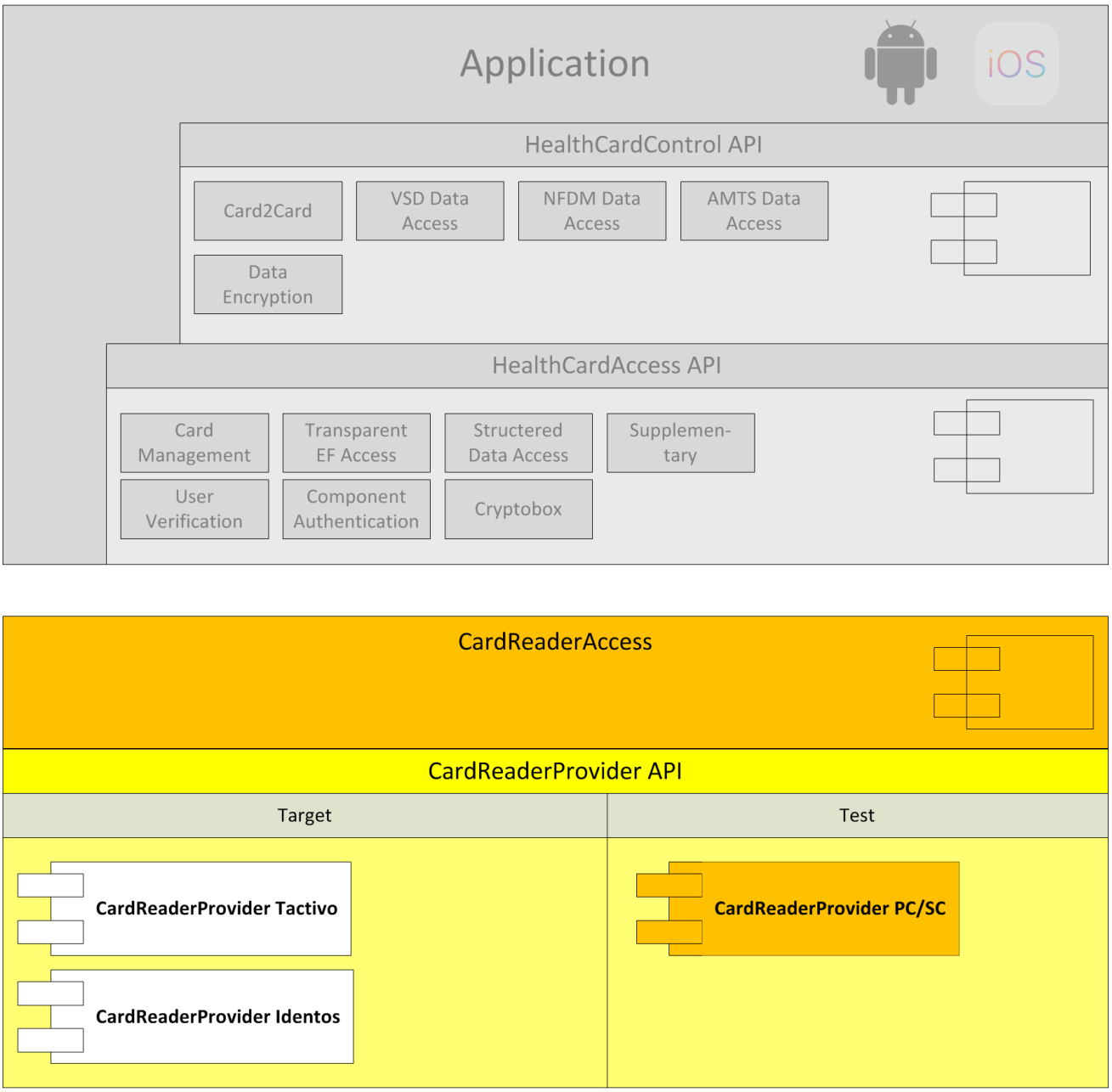
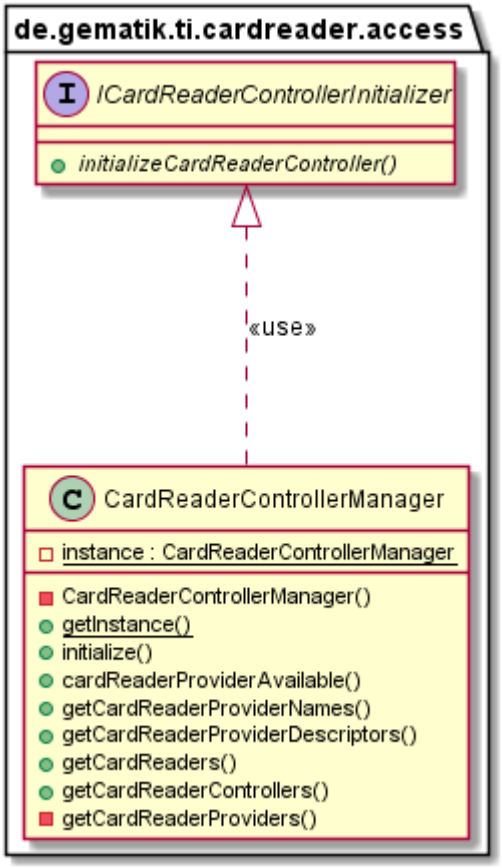



*Figure 5: CardReaderAccess API Layer*

The CardReaderAccess API provides functionalities for all card reader Service Provider discovers by ServiceLoader. CardReaderAccess API implements the ServiceLoader for card reader Service Provider. This has the role of discovering and loading implementations lazily.

## 4.2. Overview




*Class diagram 14: CardReaderAccess API*

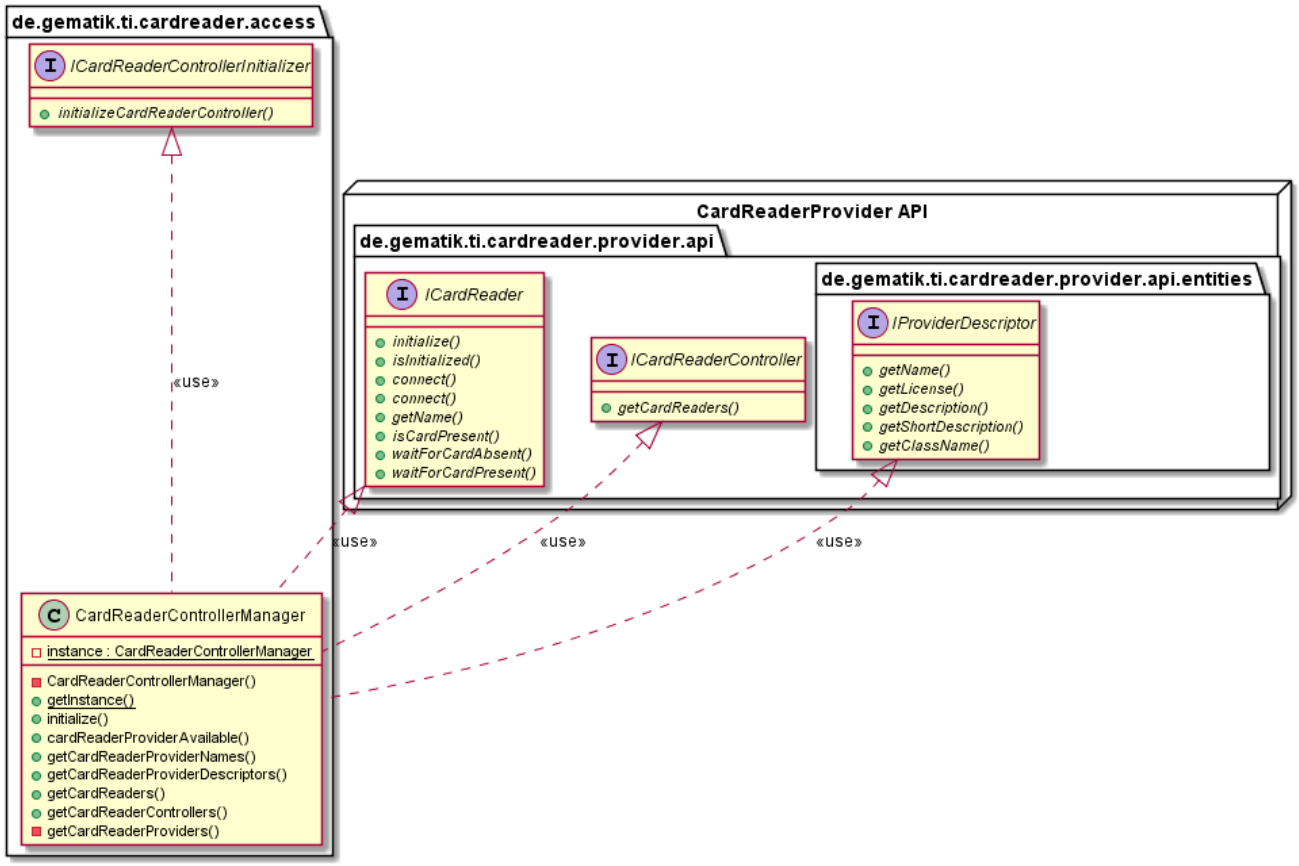
### 4.2.1. CardReader Events

If an application or library user would receive information's about card reader connection or disconnection must the receiver subscribe to EventBus for Event `CardReaderConnectedEvent` and `CardReaderDisconnectedEvent`.

For usage see CardReaderConnectionEvents

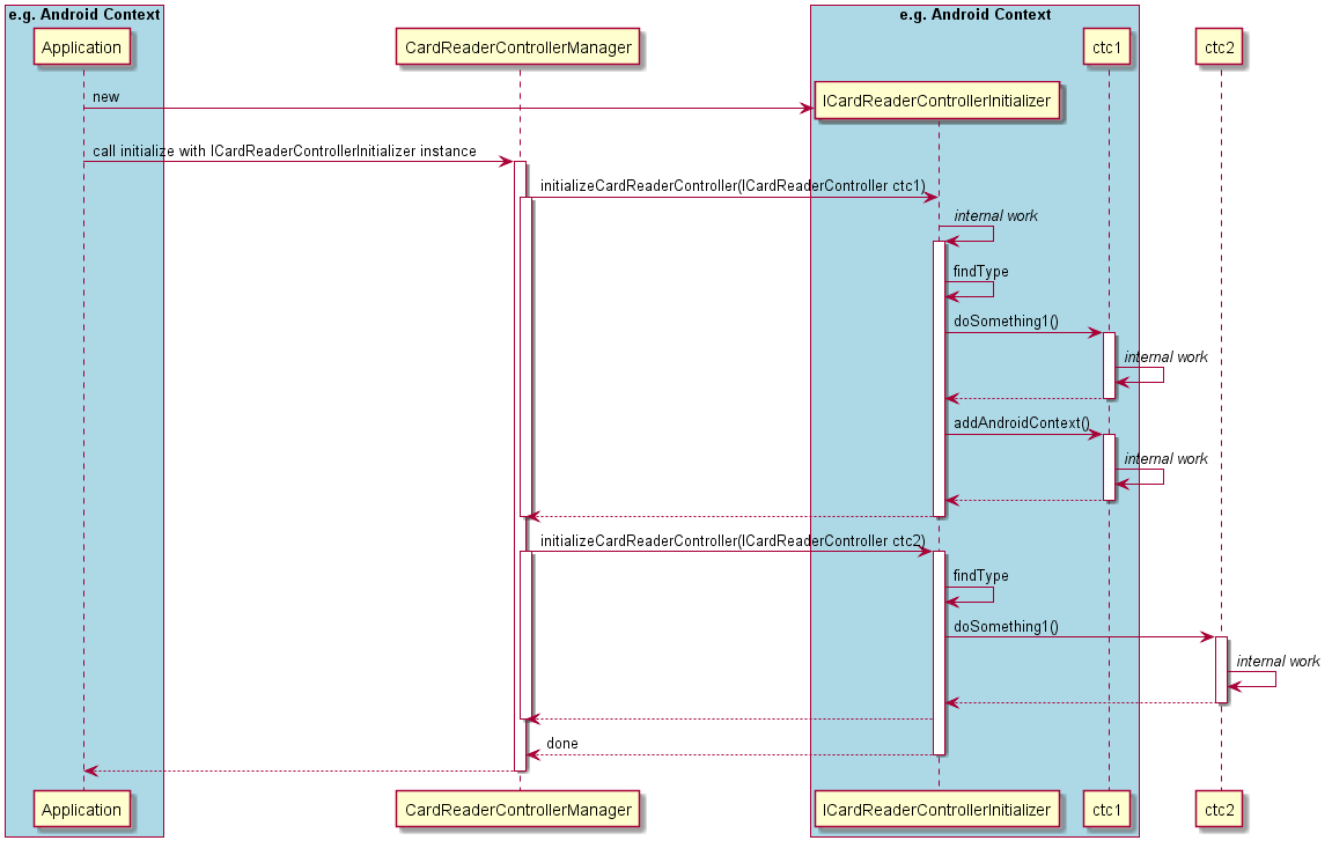
### 4.2.2. CardReaderControllerManager

This manager object provides functionalities over all CardReaderController and CardReaderProviders to request all available card readers and information's about CardReaderProviders (e.g. licence, description). Furthermore is this manager the service loader for CardReaderProviders in your application.




*Class diagram 15: CardReaderControllerManager*

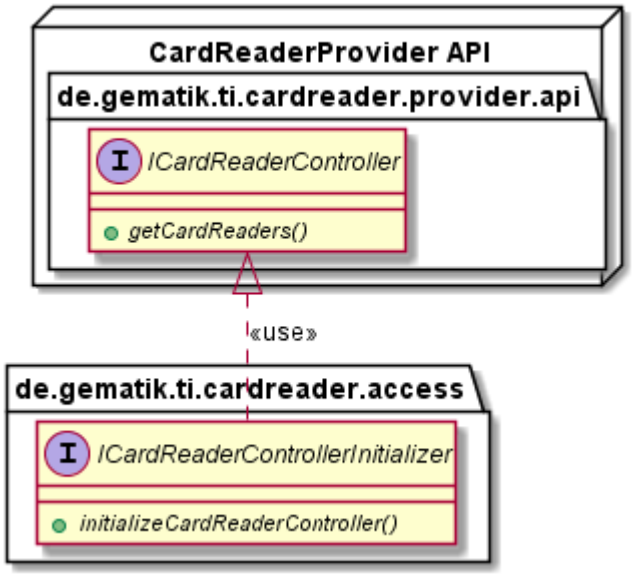
The special and optional functionality is the initializer method for CardReaderController. The initialize method will call with implementation object of [ICardReaderControllerInitializer]. The ICardReaderControllerInitializer will call from CardReaderControllerManager for each found [ICardReaderController] and the implementation of [ICardReaderControllerInitializer] will decides for which [ICardReaderController] is something todo. This methodology makes it possible to inject the e.g. android application context to service provider from higher layer applications without pass-through each layer. The following sequence diagram describe the workflow for initialization on ctc1 (represents an android specific ICardReaderController implementation) and ctc2 (plain java implementation e.g. pcsc ICardReaderController implementation). The [ICardReaderControllerInitializer] decides the need of android context.




*Sequence diagram 1: Initializing sequence for ICardReaderController with ICardReaderControllerInitializer*

### 4.2.3. CardReaderControllerInitializer

The implementing class for ´ICardReaderControllerInitializer´ interface will initialize CardReaderController e.g. inject android application context or prepare environment.




*Class diagram 16: CardReaderControllerInitializer*

## 4.3. Getting Started

### 4.3.1. Build setup

To use CardReaderAccess API library in a project, you need just to include following dependency:

*Listing 5. Gradle dependency settings to use CardReaderAccess API library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'cardreader.access', version: '1.+'
}
```

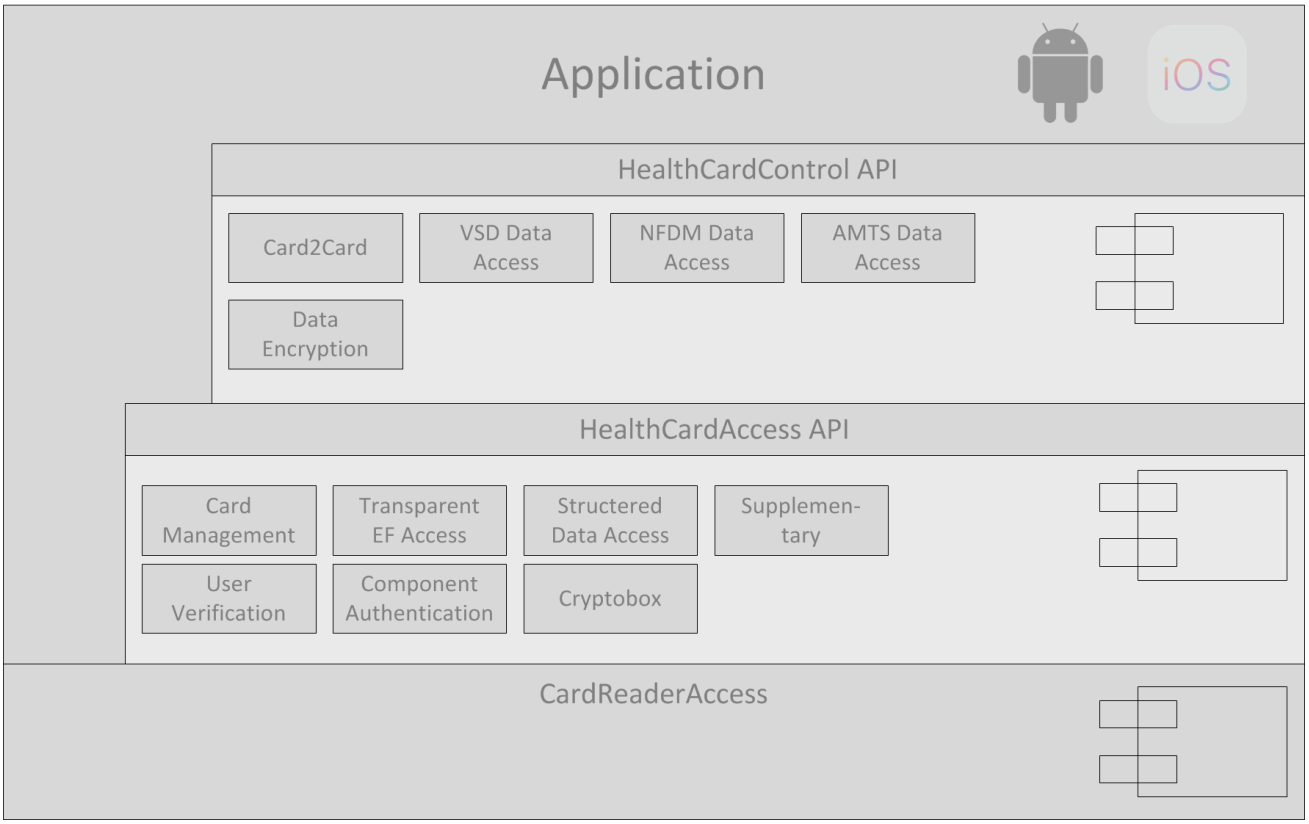
*Listing 6. Maven dependency settings to use CardReaderAccess API library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>cardreader.access</artifactId>
        <version>1.+</version>
    </dependency>
</dependencies>
```

# Chapter 5. CardReaderProvider API

## 5.1. Introduction

The CardReaderProvider API define the general communication interface between application and HealthCardReader to abstract the specific implementation for each hardware card reader. This part describes the usage of low level CardReaderProvider API in order to use CardReader in your application.



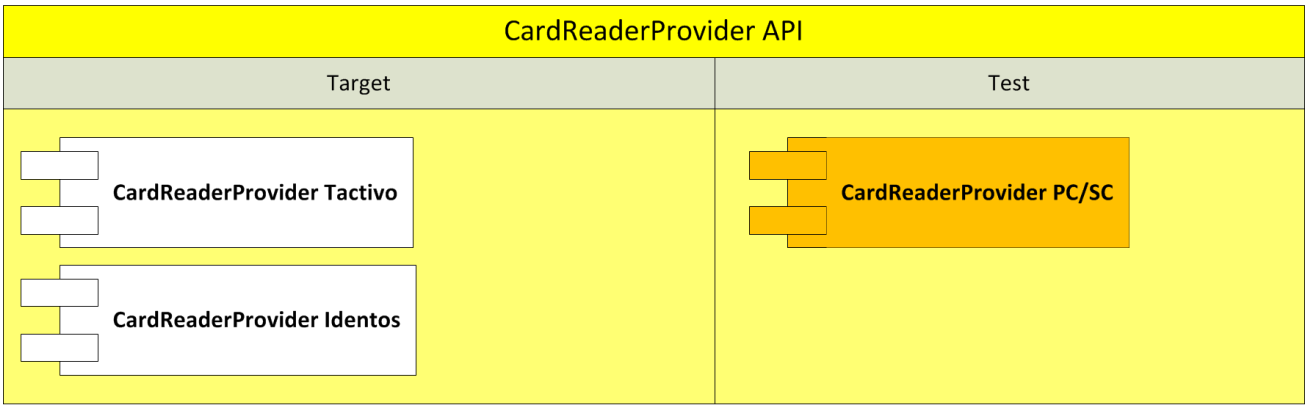




*Figure 6: CardReaderProvider API Layer*

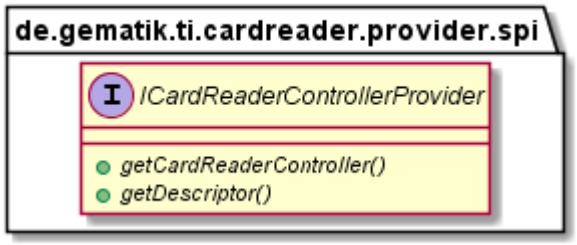
The CardReaderProvider for each specific hardware are implemented as Java Service Provider and would load dynamically by ServiceLoader implemented in CardReaderAccess.

## 5.2. Overview

This library provides a general (Java SPI) API for abstracting card reader specific implementations. On mobile devices, each card reader has its own driver and controller implemention provided by card reader manufacturer and thus needs an own provider implementation adapting it to this generic API. Card reader provider implementations are loaded during runtime by Java's Service Provider (SPI) mechanism. The necessary artifacts to implement such a service provider are described in following chapters.

### 5.2.1. Service Provider

The entry point for the ServiceLoader is ICardReaderControllerProvider Interface. This Interface returns the specific CardReaderController implementation and a Descriptor class




*Class diagram 17: ICardTerminalControllerProvider*

The specific cardreader.provider needs a descriptor within folder `YOUR.PROVIDER\src\main\resources\META-INF\services` with filename `de.gematik.ti.cardreader.provider.spi.ICardReaderControllerProvider` and the content of the package and class which implements the service provider interface e.g. `de.gematik.ti.cardreader.provider.pcsc.control.CardReaderProvider`

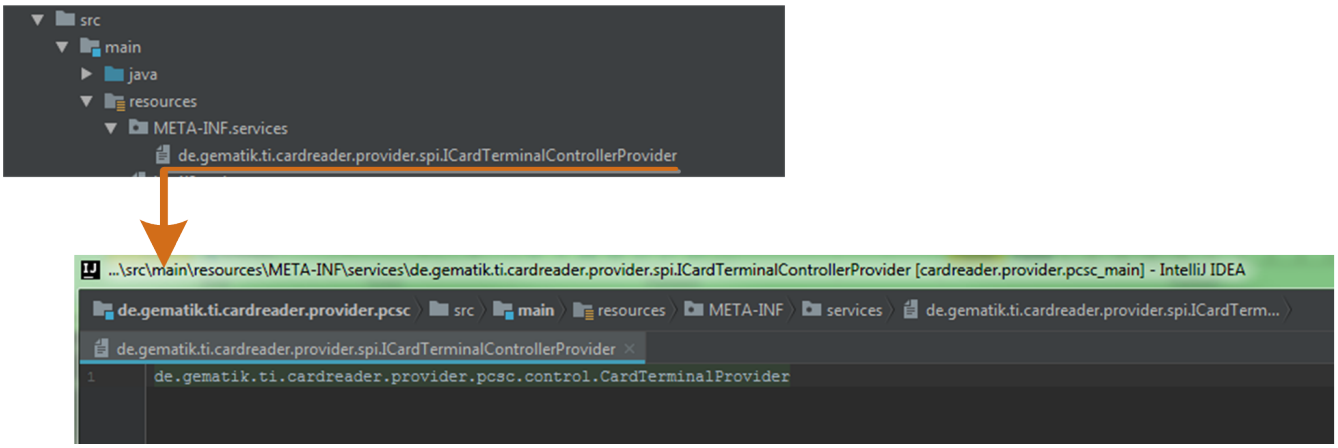
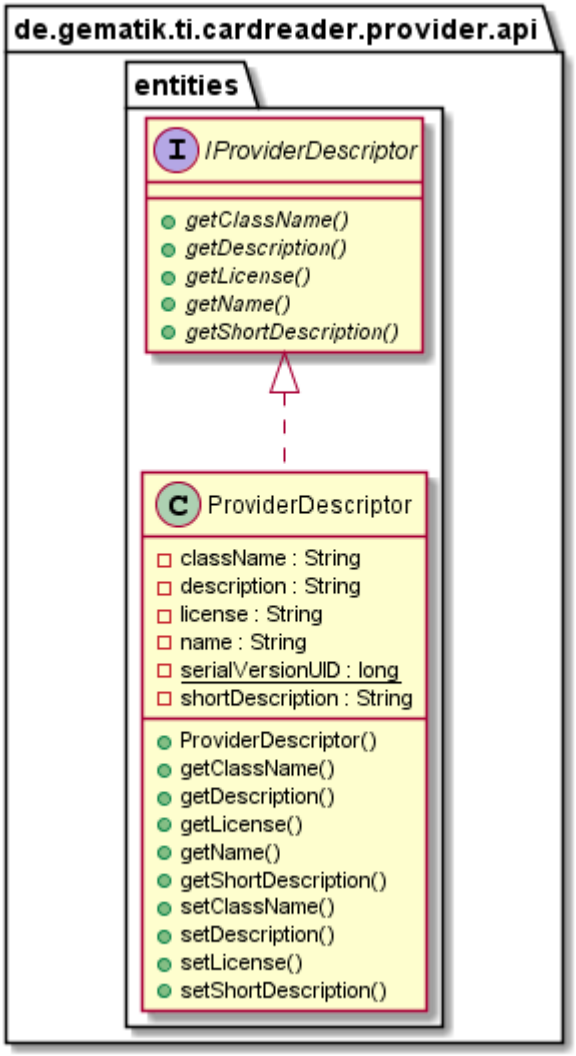



*Figure 7: File META-INF services*

### 5.2.2. Descriptor

Each implementation of cardreader provider contains an descriptor which contains information about name, licence, provider descriptions, etc. This class implements the `IProviderDescriptor` interface. The CardReaderProvider API delivers a default implementation with the class

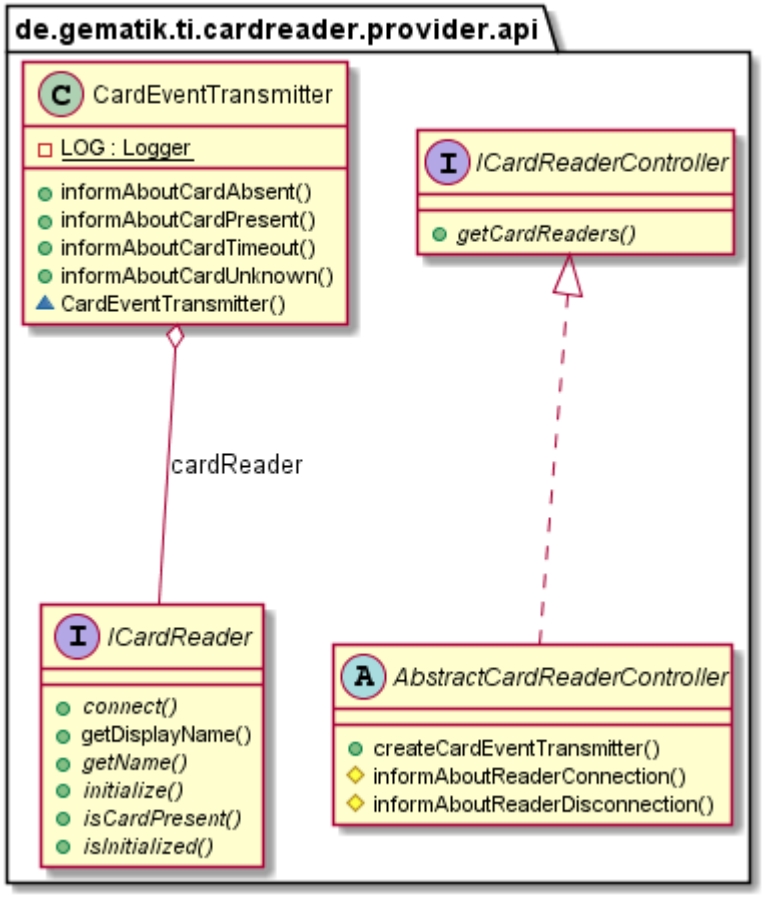
`ProviderDesciptor`.




*Class diagram 18: ProviderDescriptor*

### 5.2.3. CardReaderController

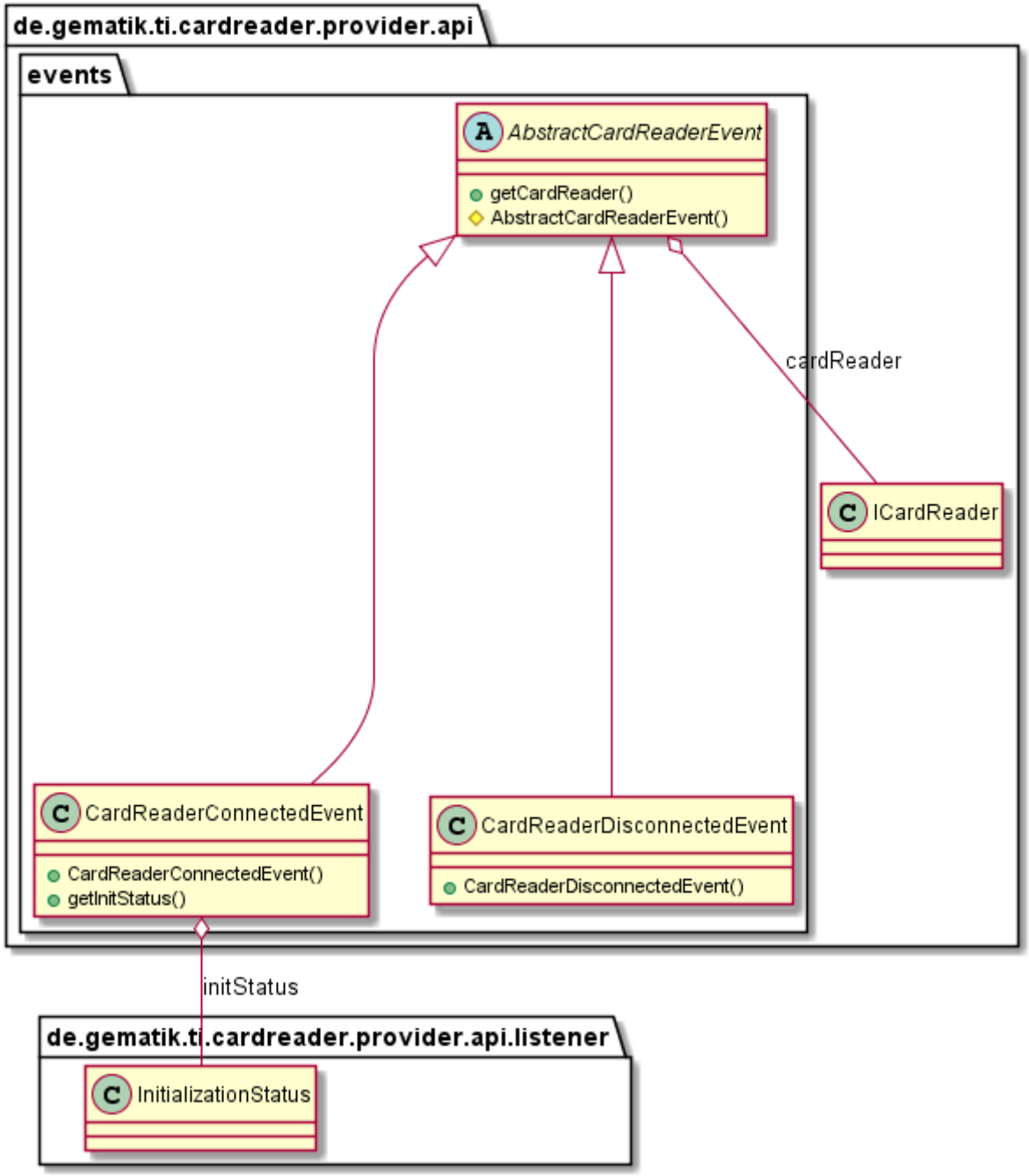
The cardreader provider needs an implementation class of `ICardReaderController` interface. This class handles the card reader for the higher layer application. The application could request available CardReader. CardReaderProvider API provides a default implementation of `ICardReaderController` interface as abstract class `AbstractCardReaderController` for ease of use. The abstract class handles event bus communication and provides methods to inform registered subscribers about card reader changes.




*Class diagram 19: ICardReaderController*

### 5.2.4. CardReaderConnectionEvents

Each class which should send connection and disconnection events of card reader must register itself as Subscriber at EventBus. The CardReaderController informs about new card reader and disconnected card reader by sending appropriate events on EventBus. The information about new card reader contains additional information about `InitializationStatus`. This information is important for higher layers to decide on the initialisation procedure e.g. request permissions.




*Class diagram 20: Card Reader connection events*

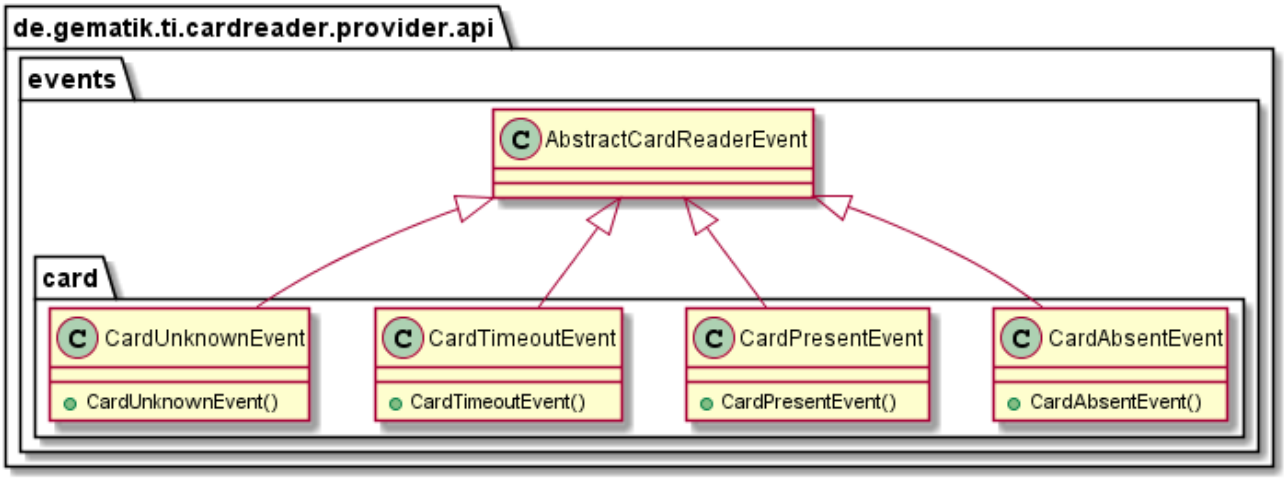
Each class interested in connection events can register a `CardReaderConnectedEvent` and `CardReaderDisconnectedEvent` receiving method at Eventbus.

*Listing 7. CardReaderConnectedEvent Example*

```
@Subscribe
public void cardReaderConnected(final CardReaderConnectedEvent connectedEvent) {
    // Do Something
}

public void registerOnEventBus() {
    EventBus.getDefault().register(this);
}

public void unregisterOnEventBus() {
    EventBus.getDefault().unregister(this);
}
```

### 5.2.5. CardEvents

Each class which should inform about card events from card reader must register as subscriber at EventBus. The CardEventTransmitter informs about events on EventBus.




*Class diagram 21: Card Events*

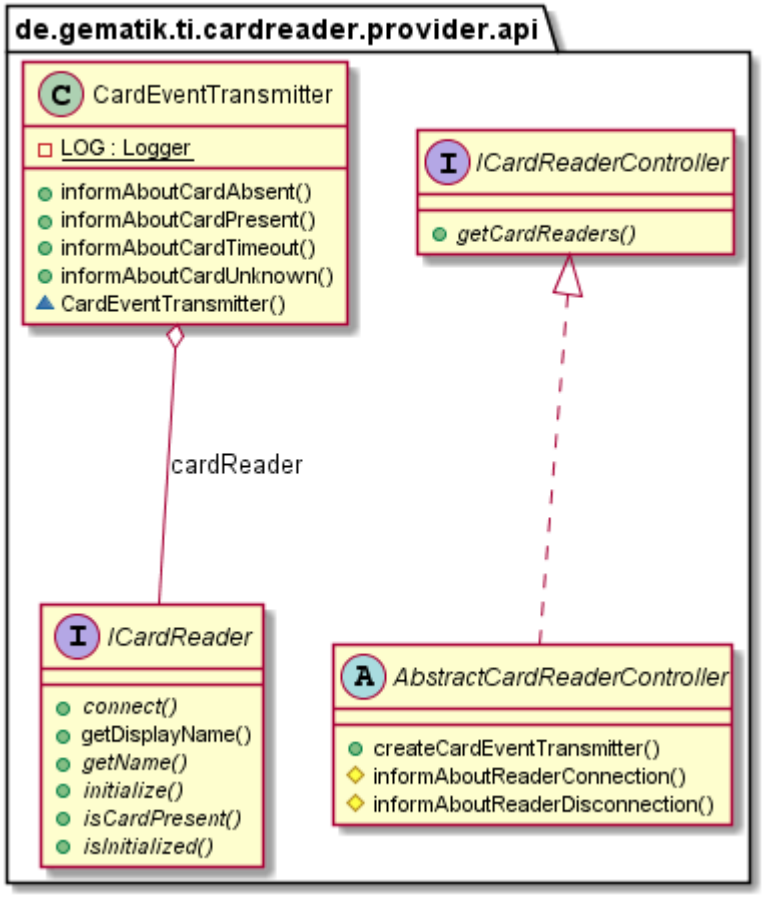
Eeach interested class can register a `CardPresentEvent`, `CardAbsentEvent`, ... receiving method at Eventbus.

*Listing 8. CardEvent Example*

```
public void registerOnEventBus() {
    EventBus.getDefault().register(this);
}

@Subscribe
public void cardConnected(final CardPresentEvent connectedEvent) {
    connectedCards += 1;
    logger.debug("cardConnected, Count: " + connectedCards);
}

@Subscribe
public void cardDisconnected(final CardAbsentEvent disconnectedEvent) {
    connectedCards -= 1;
    logger.debug("cardDisconnected, Count: " + connectedCards);

}
```

### 5.2.6. CardEventTransmitter

Each class which should inform about card presence events must register itself as subscriber at EventBus. The CardReaderController creates a CardEventTransmitter for each card reader. This transmitter informs about new cards, absent cards, timeouts and unknown cards with appropriate events on EventBus.




*Class diagram 22: Card connection events*

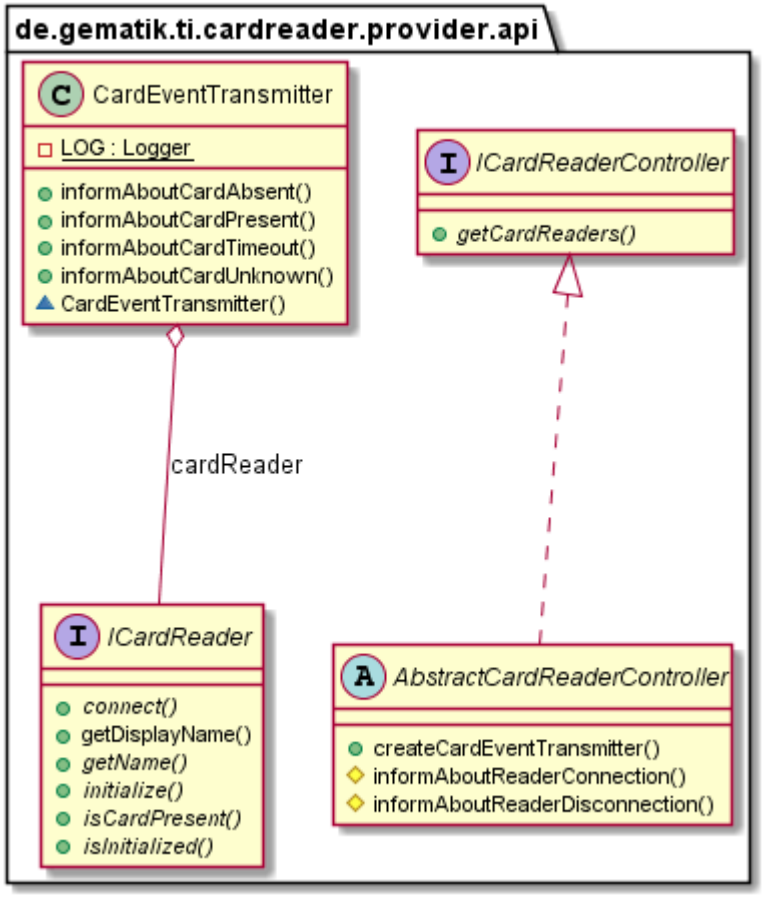
Example how to send card events with `CardEventTransmitter`

*Listing 9. CardEventTransmitter Example*

```
private final ICardReader reader = Mockito.mock(ICardReader.class);

@Test
public void sendCardEvent() {
    /* abstractCardReaderController; */ // The Provider specific Controller
implementation
    CardEventTransmitter cardEventTransmitter =
abstractCardReaderController.createCardEventTransmitter(reader);
    cardEventTransmitter.informAboutCardPresent();
}
```

### 5.2.7. CardReader

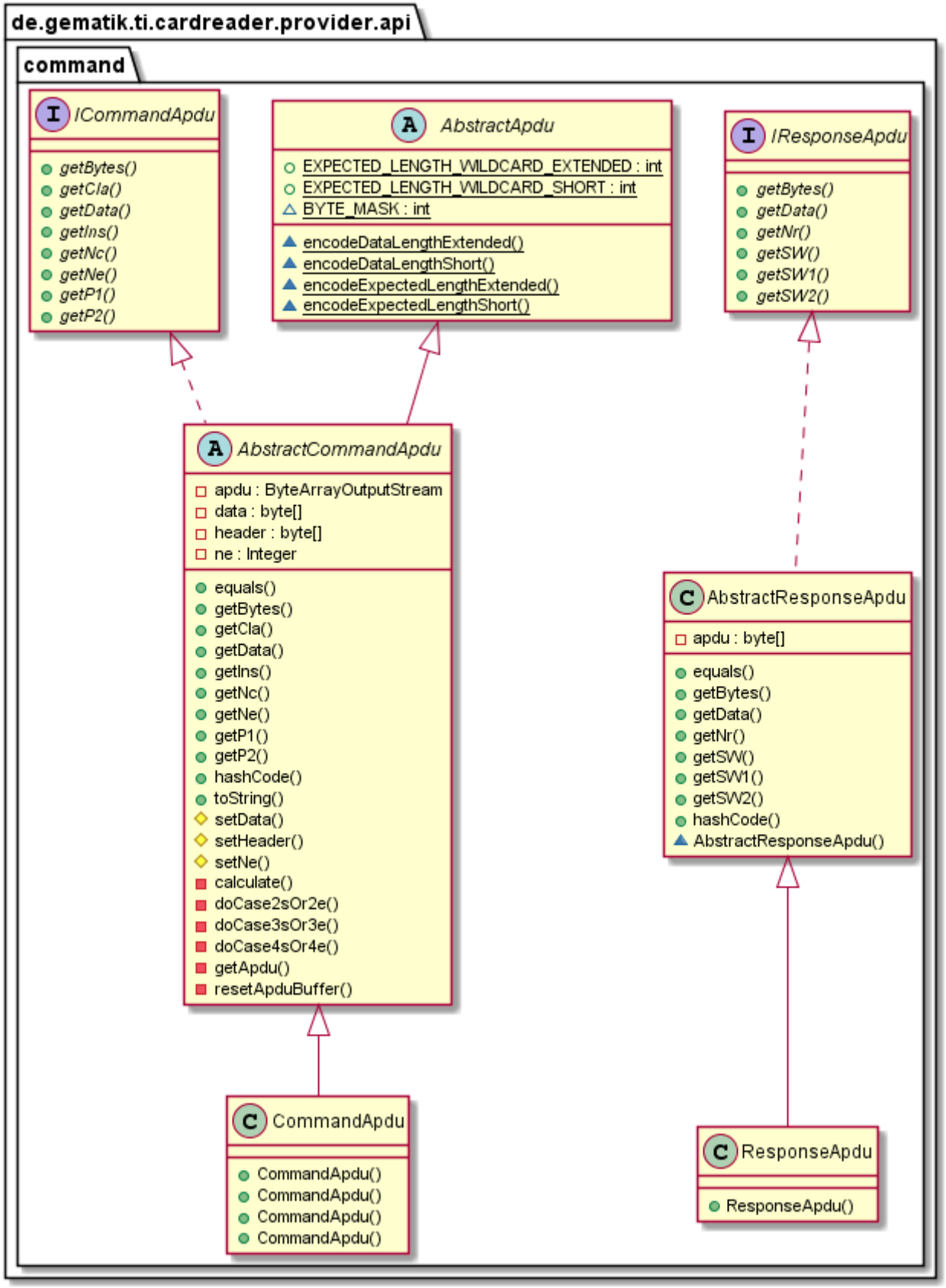
The physical card reader is represented by a class implementing `ICardReader` interface. This class handles card reader's initialisation status and card.




*Class diagram 23: ICardReader*

### 5.2.8. Card Commands

The Api provides an general structure of a card command and responses, so called APDU. The detailed implementation of all specific commands healtcards can process is done in a higher layers library.




*Class diagram 24: Card Commands*

## 5.3. Getting Started

### 5.3.1. Build setup

To use CardReaderProvider API library in a project, you need just to include following dependency:

*Listing 10. Gradle dependency settings to use CardReaderProvider API library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'cardreader.provider.api', version:
'1.+'
}
```

*Listing 11. Maven dependency settings to use CardReaderProvider API library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>cardreader.provider.api</artifactId>
        <version>1.+</version>
    </dependency>
</dependencies>
```

# Chapter 6. Card Reader Providers

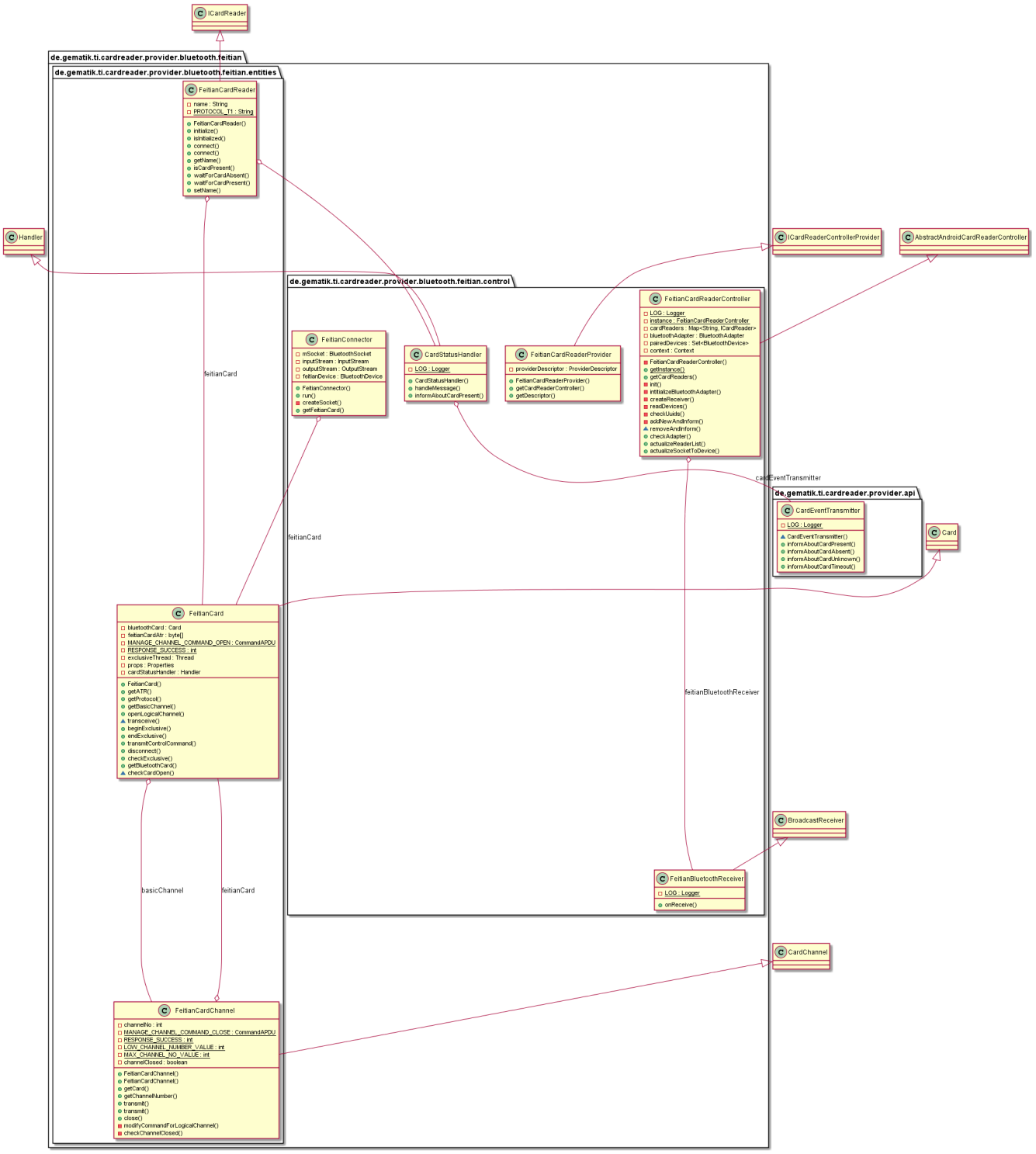
This Chapter describe the Card Reader Providers implemented by gematik - Gesellschaft für Telematikanwendungen der Gesundheitskarte mbH

## 6.1. Feitian Bluetooth CardReaderProvider

## 6.2. Introduction

This part describes the usage of low level CardReaderProvider for feitian bluetooth CardReader in your application.

## 6.3. Overview




*Class diagram 25: FeitianBluetoothCardReaderProvider*

### 6.3.1. Integration

The Feitian Bluetooth CardReaderProvider needs a descriptor behind `YOUR.PROVIDER\src\main\resources\META-INF\services` with filename `de.gematik.ti.cardreader.provider.spi.ICardReaderControllerProvider` and the content of the package and class which implements the service provider interface `de.gematik.ti.cardreader.provider.bluetooth.feithian.control.FeitianCardReaderProvider`.

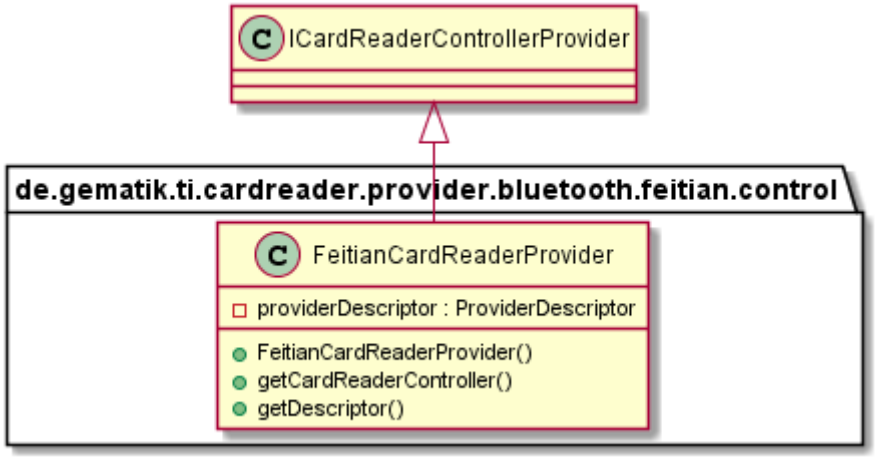
## 6.4. Hardware

The integrated Feitian Combo SDK supports the following FEITAN Readers: bR301,iR301,bR301BLE,bR500.

## 6.5. Control

### 6.5.1. FeitianCardReaderProvider

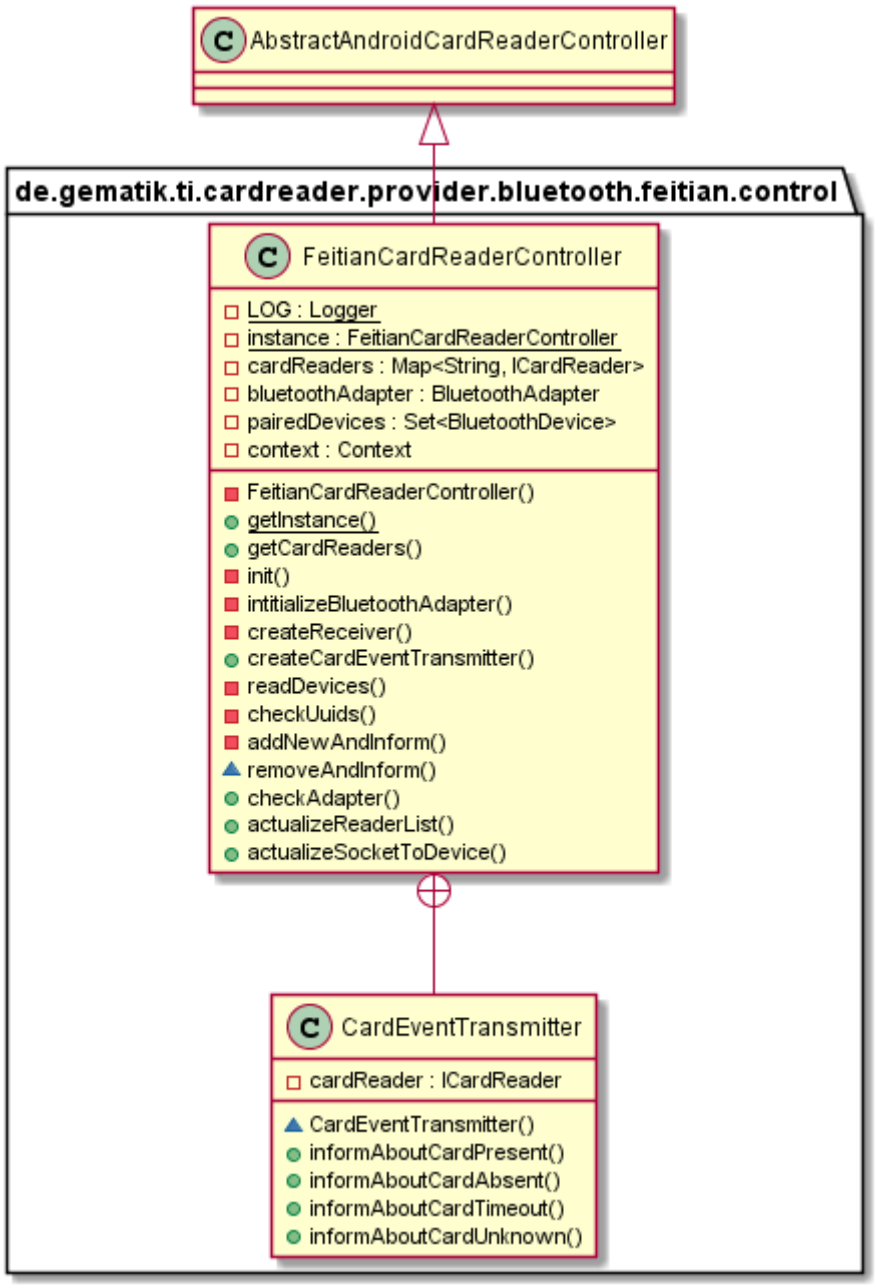
The FeitianCardReaderProvider class needs implementation of the interface 'ICardReaderControllerProvider' to handle listener and provide methods to inform connected listeners about card reader changes.




*Class diagram 26: FeitianCardReaderProvider*

### 6.5.2. FeitianCardReaderController

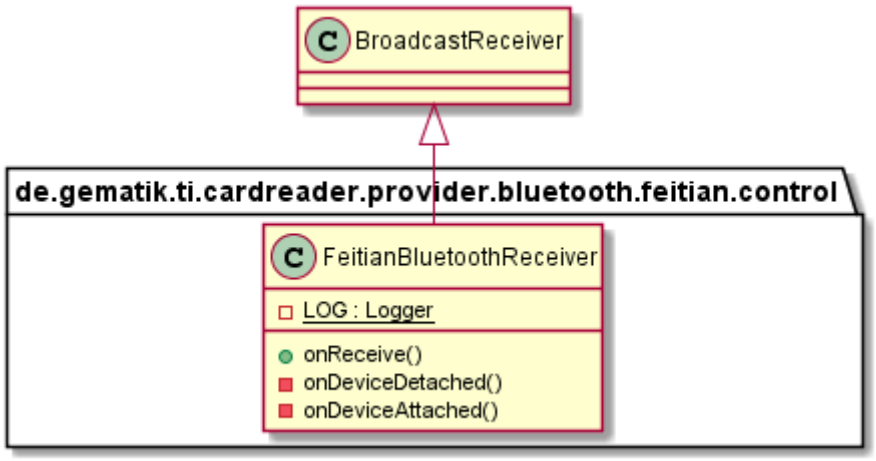
The FeitianCardReaderController class extends the abstract class 'AbstractCardReaderController' to handle necessary permissions and checking if the application context is set. Returns a list with currently connected Feitian cardReaders and informs about Reader connection and disconnection.




*Class diagram 27: FeitianCardReaderController*

### 6.5.3. FeitianBluetoothReceiver

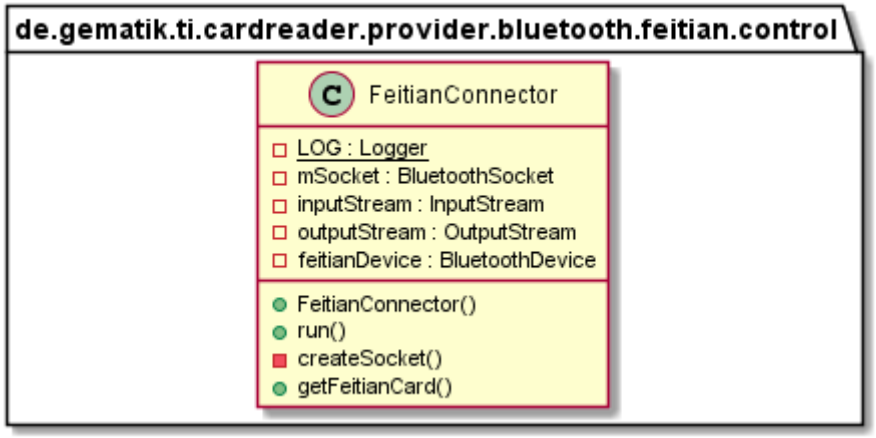
The FeitianBluetoothReceiver class extends the BroadCastReceiver class that receives and handles broadcast intents sent by {@link android.content.Context#sendBroadcast(Intent)}. The actions of the intents that are being handled are changes in the state of the bluetooth adapter (enable / change) as well as connect and disconnect a bluetooth device.




*Class diagram 28: FeitianBluetoothReceiver*

#### 6.5.4. FeitianConnector

Creates a Socket to the connected Feitian bluetooth device.

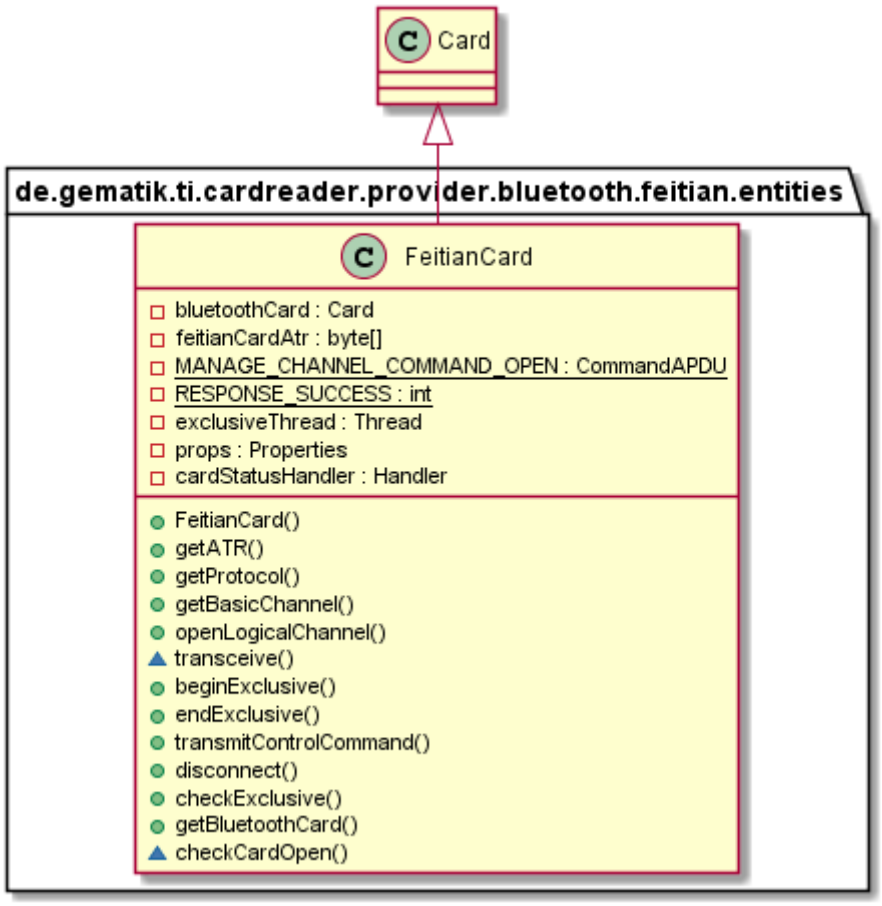



*Class diagram 29: FeitianConnector*

## 6.6. Entities

#### 6.6.1. FeitianCard

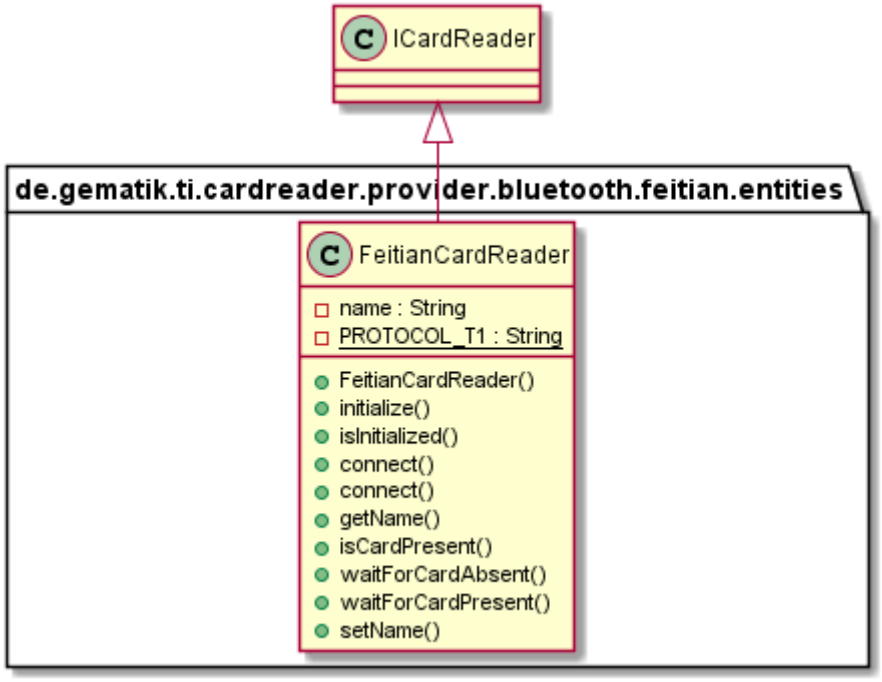
A Feithian Bluetooth Smart Card with which a connection has been established. Card objects are obtained by calling {@link FeitianConnector#run()} if a socket to the bluetooth device is connected. Works as adapter to the Feitian Bluetooth SDK.




*Class diagram 30: FeitianCard*

### 6.6.2. FeitianCardReader

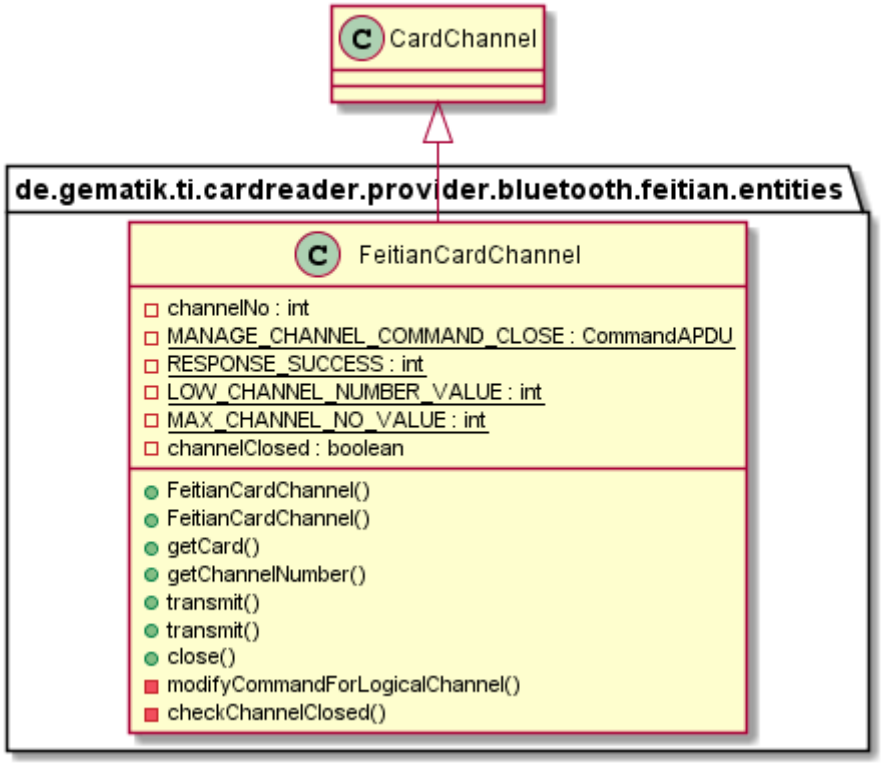
A Bluetooth Smart Card Reader, sometimes referred to as a Bluetooth Card Reader implements the Interface ICardReader.




*Class diagram 31: FeitianCardReader*

### 6.6.3. FeitianCardChannel

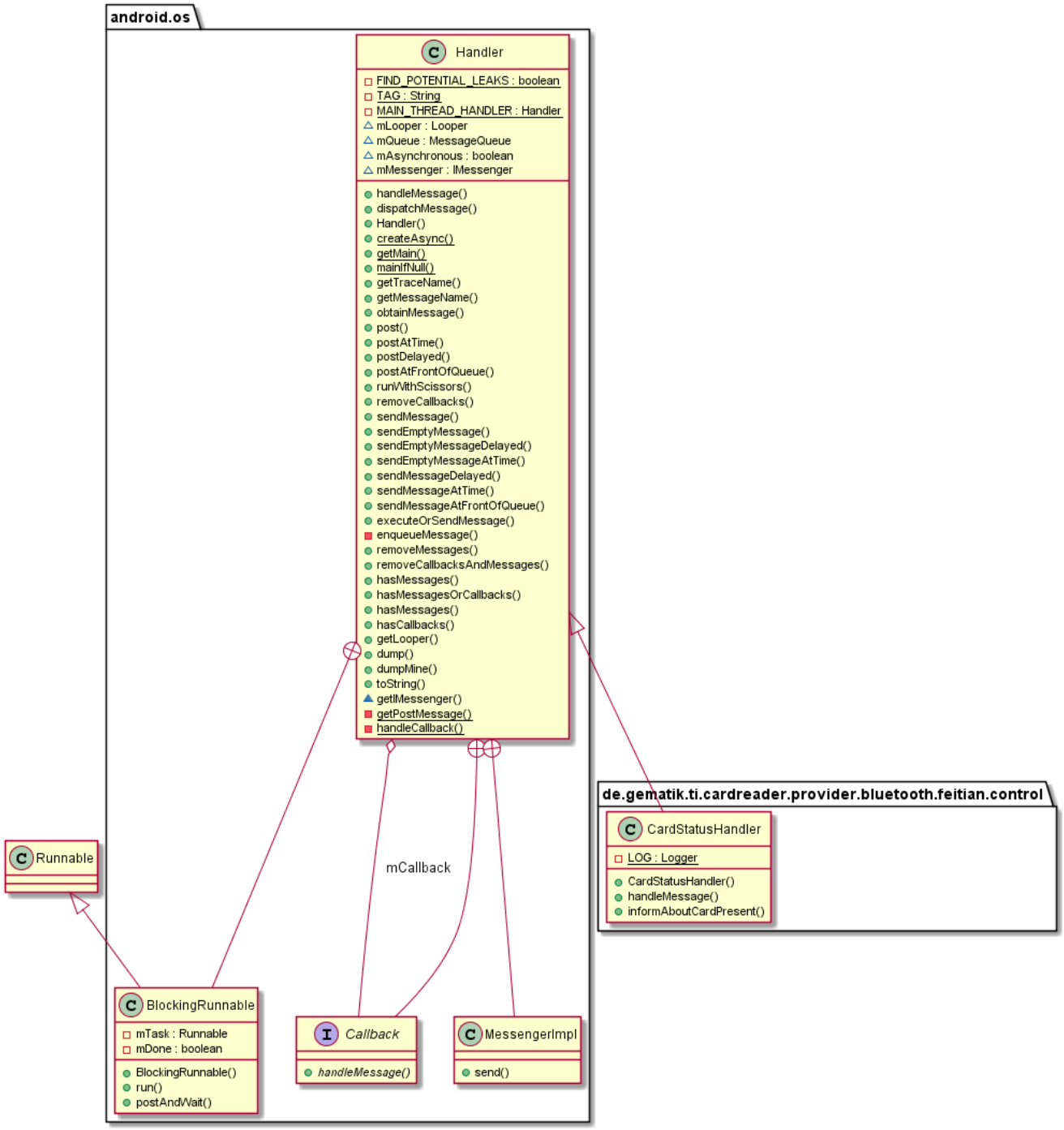
A logical channel connection to a Bluetooth Smart Card. It is used to exchange APDUs with a Smart Card using Bluetooth/CCID. A BluetoothCardChannel object can be obtained by calling the method FeitianCard.getBasicChannel or FeitianCard.openLogicalChannel




*Class diagram 32: FeitianCardChannel*

### 6.6.4. CardStatusHandler

Each FeitianCardReader needs an Handler for Card Event. This Handler compute the card reader card events and transmit the suited events to EventBus subscriber.




*Class diagram 33: CardStatusHandler*

## 6.7. Getting Started

### 6.7.1. Build setup

To use CardReaderProvider for feitian bluetooth CardReader in a project, you need just to include following dependency:

*Listing 12. Gradle dependency settings to use CardReaderProvider for feitian bluetooth CardReader library*

```
dependencies {
    implementation group: 'de.gematik.ti', name:
'cardreader.provider.bluetooth.feitian', version: '1.+'
}
```

*Listing 13. Maven dependency settings to use CardReaderProvider for feitian bluetooth CardReader  library*

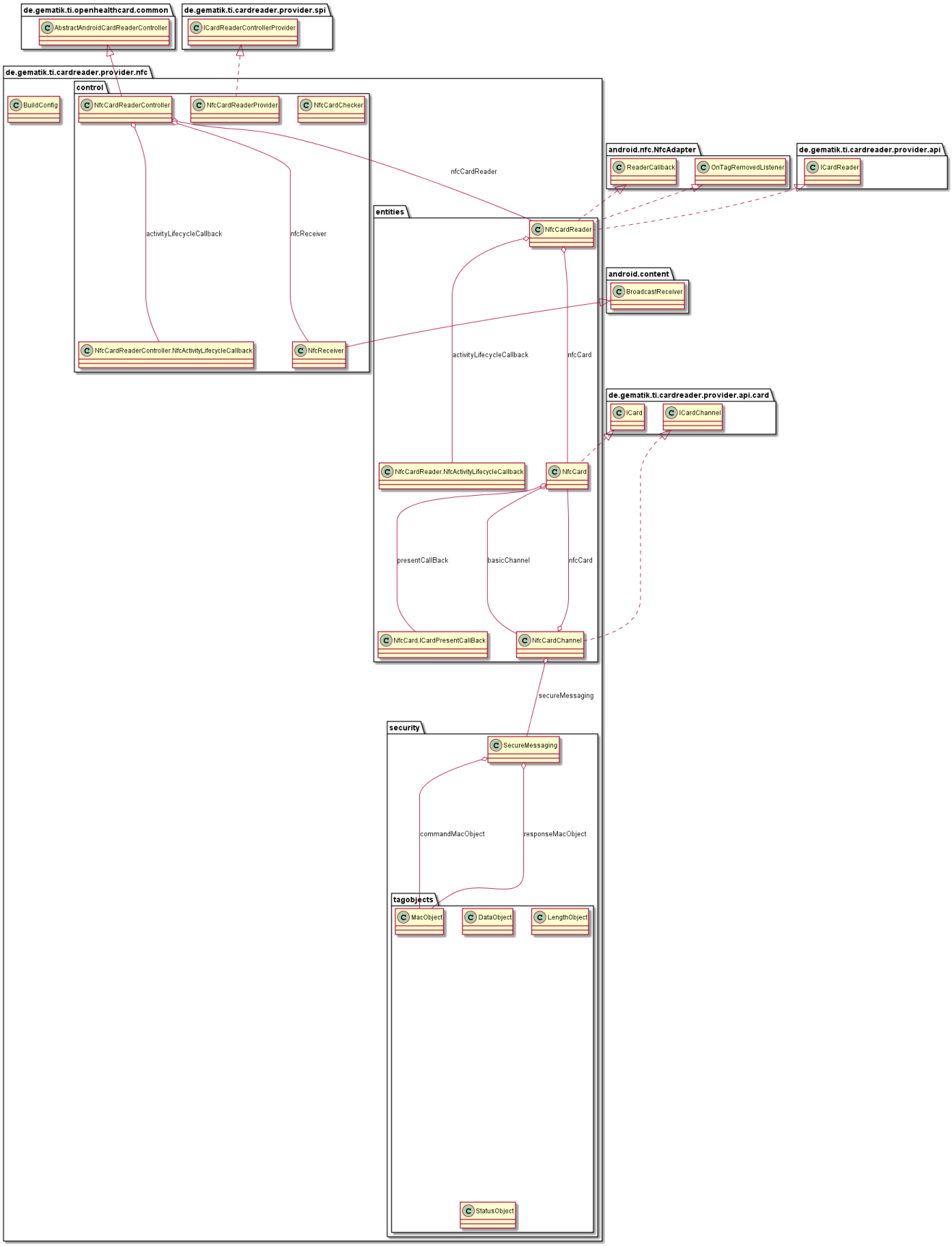
```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>cardreader.provider.bluetooth.feitian</artifactId>
        <version>1.+</version>
    </dependency>
</dependencies>
```

# 6.8. NFC CardReaderProvider

## 6.8.1. Introduction

This part describes the usage of low level CardReaderProvider for NFC CardReader in your application.

## 6.8.2. Overview




*Class diagram 34: NfcCardReaderProvider*

### 6.8.2.1. Integration

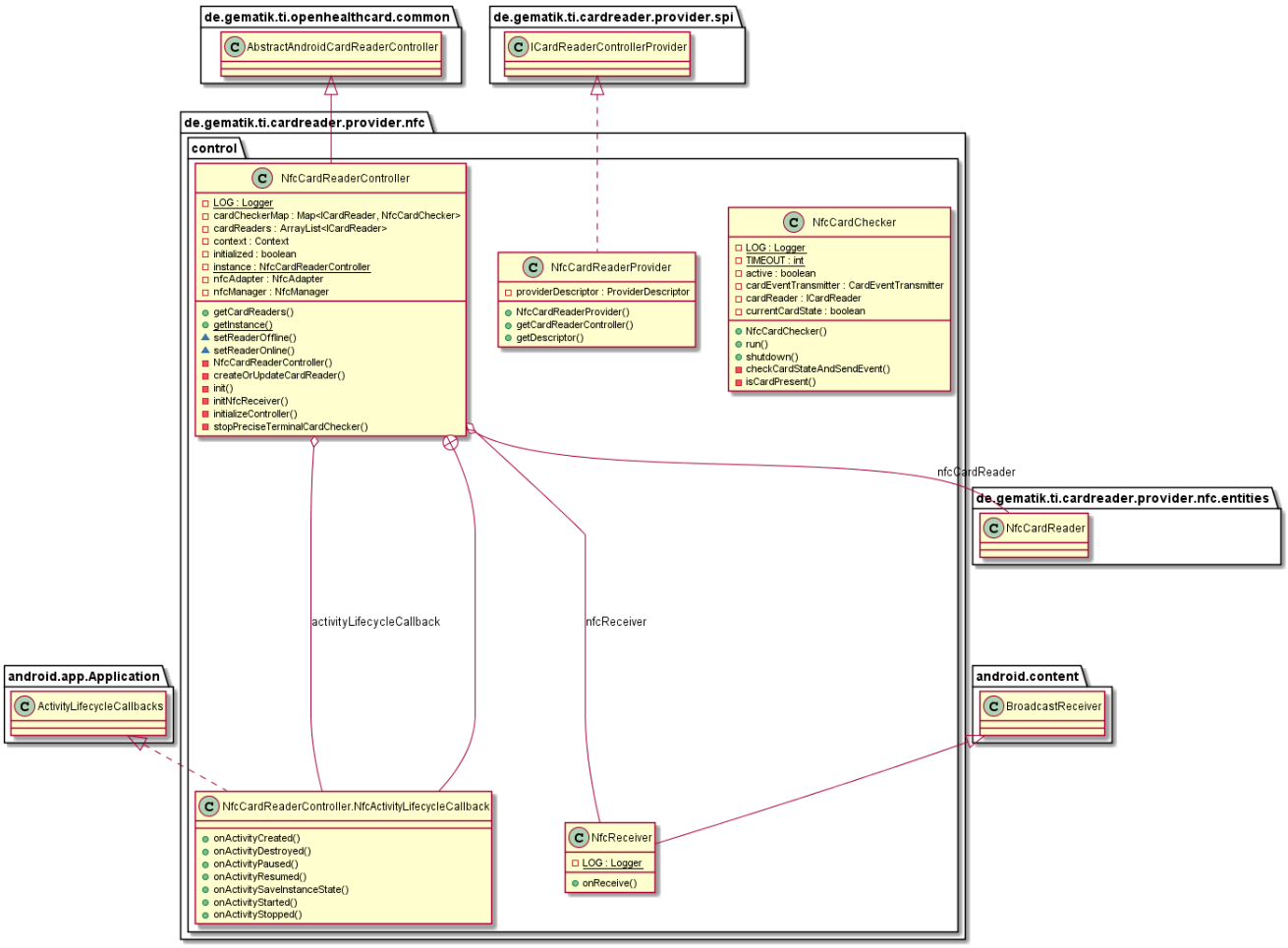
The NFC CardReaderProvider needs a descriptor behind `YOUR.PROVIDER\src\main\resources\META-INF\services` with filename `de.gematik.ti.cardreader.provider.spi.ICardReaderControllerProvider` and the content of the package and class which implements the service provider interface

`de.gematik.ti.cardreader.provider.usb.nfc.control.NfcCardReaderProvider`.

### 6.8.3. Hardware

The NFC CardReaderProvider supports the integrating of smart card functionality using NFC.

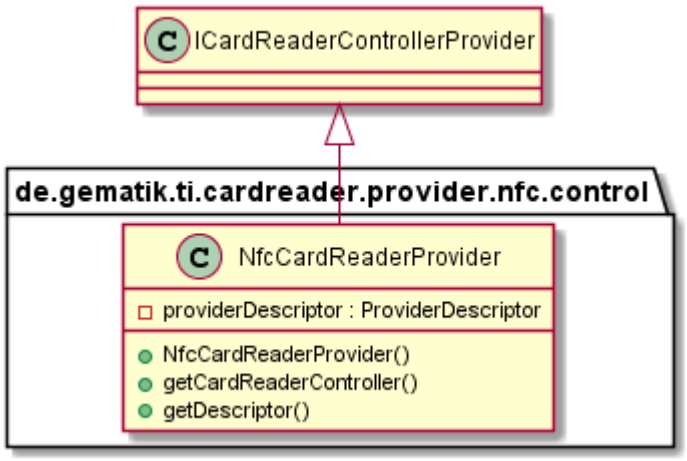
### 6.8.4. Control




*Class diagram 35: Nfc Control*

#### 6.8.4.1. NfcCardReaderProvider

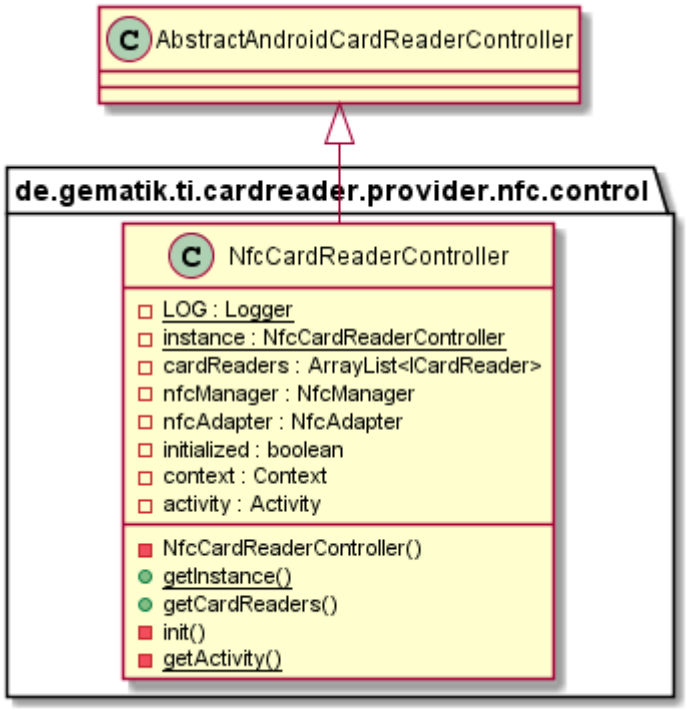
The NfcCardReaderProvider class needs implementation of the interface 'ICardReaderControllerProvider' to handle listener and provide methods to inform connected listeners about card reader changes.




*Class diagram 36: NfcCardReaderProvider*

#### 6.8.4.2. NfcCardReaderController

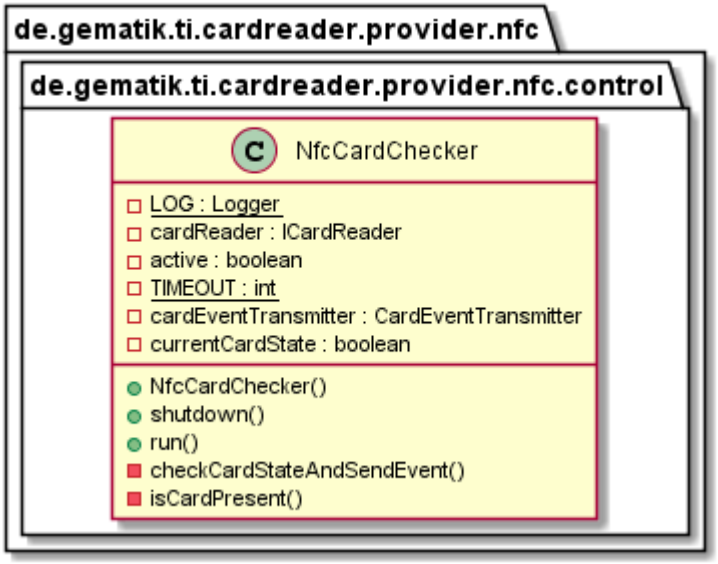
The NfcCardReaderController class extends the abstract class 'AbstractCardReaderController' to handle necessary permissions and checking if the application context is set. Returns a list with currently connected Tactivo cardReaders and informs about reader connection and disconnection.




*Class diagram 37: NfcCardReaderController*

#### 6.8.4.3. NfcReceiver

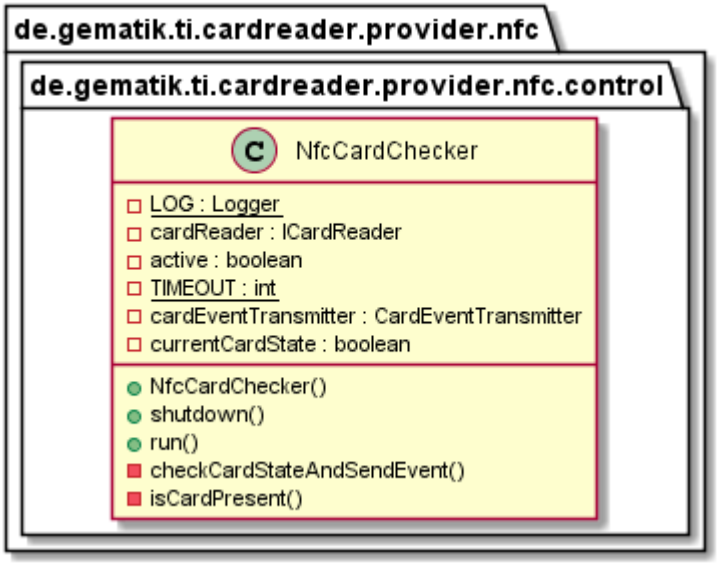
The NfcReceiver class extends the BroadCastReceiver class that receives and handles broadcast intents sent by {@link android.content.Context#sendBroadcast(Intent)}. The actions of the intents that are being handled are changes in the state of the NFC adapter.




*Class diagram 38: NfcReceiver*

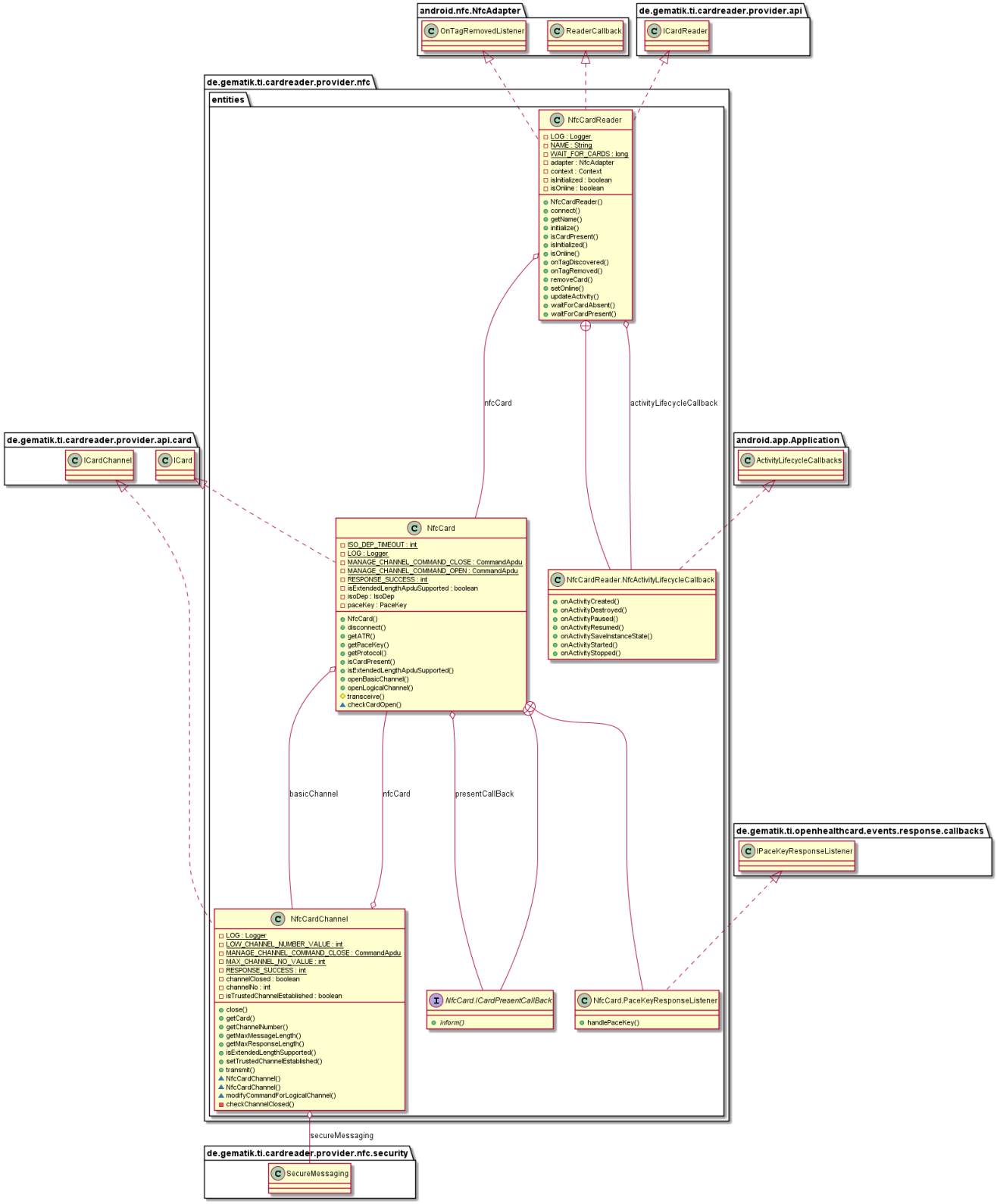
#### 6.8.4.4. NfcCardChecker

The NfcCardChecker would automatically started for each connected NfcCardReader to monitor the current card status. This checker send Events on EventBus for each present or absent card. For triggering this changed would use the SmartCardIo methods `waitForCardAbsent` and `waitForCardPresent`.




*Class diagram 39: NfcCardChecker*

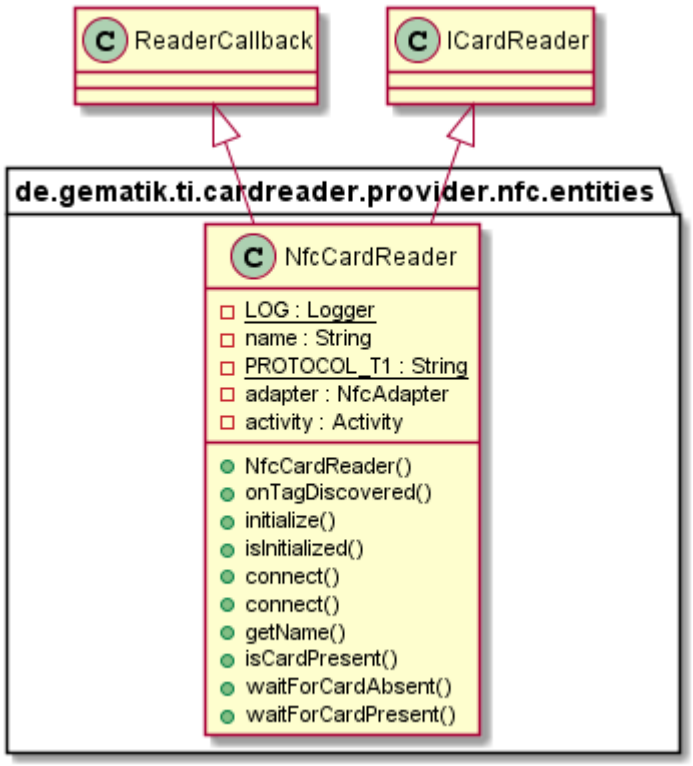
### 6.8.5. Entities




*Class diagram 40: Entities*

### 6.8.5.1. NfcCardReader

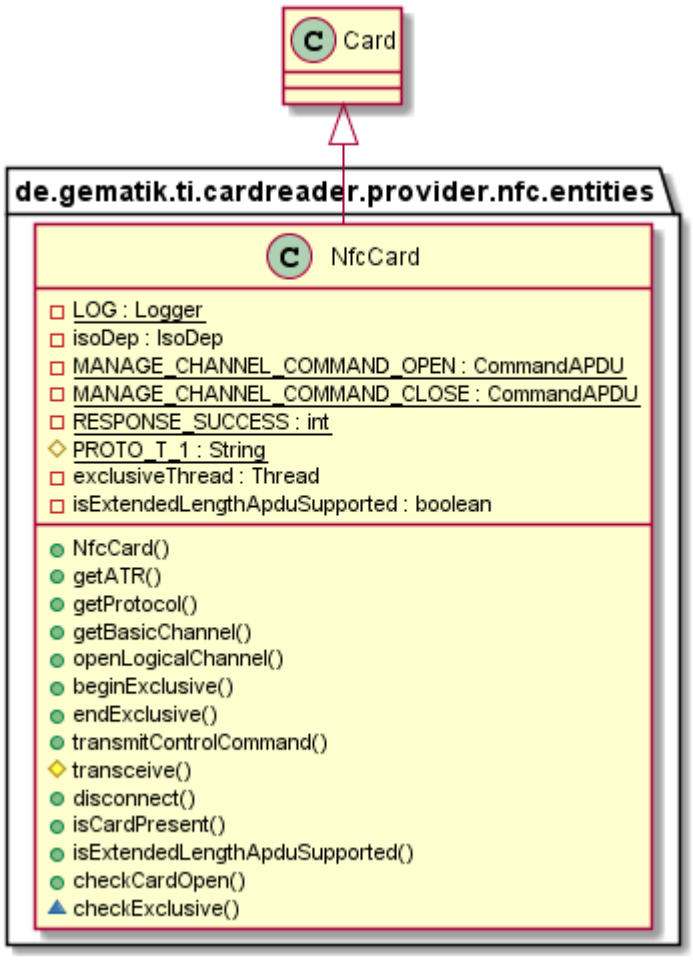
The NFC Smart Card Reader implements the Interface ICardReader. Works as Adapter to the Android NfcManager and uses the Android ReaderCallback from Android NfcAdapter to detect NFC SmartCards.




*Class diagram 41: NfcCardReader*

#### 6.8.5.2. NfcCard

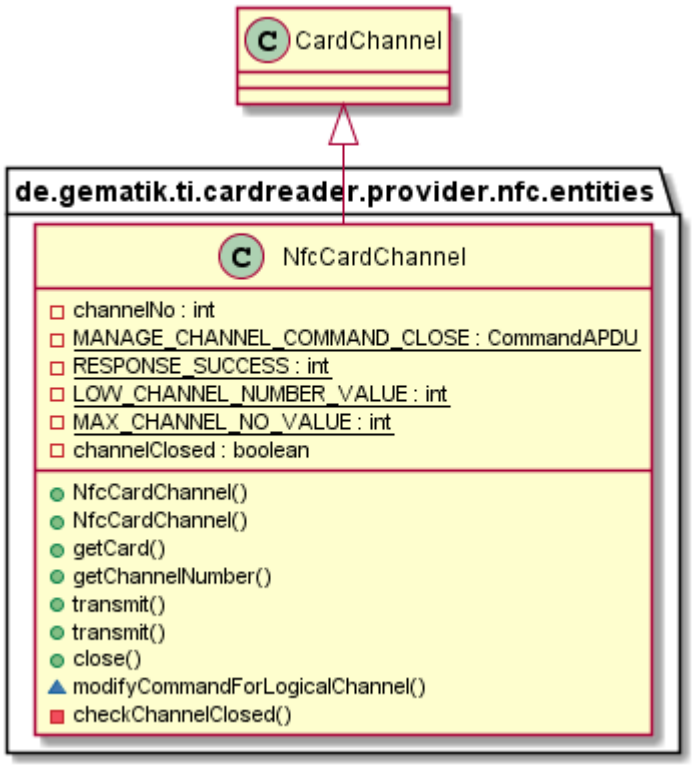
A Smart Card with which a connection has been established. NfcCard extends the abstract class Card.




*Class diagram 42: NfcCard*

#### 6.8.5.3. NfcCardChannel

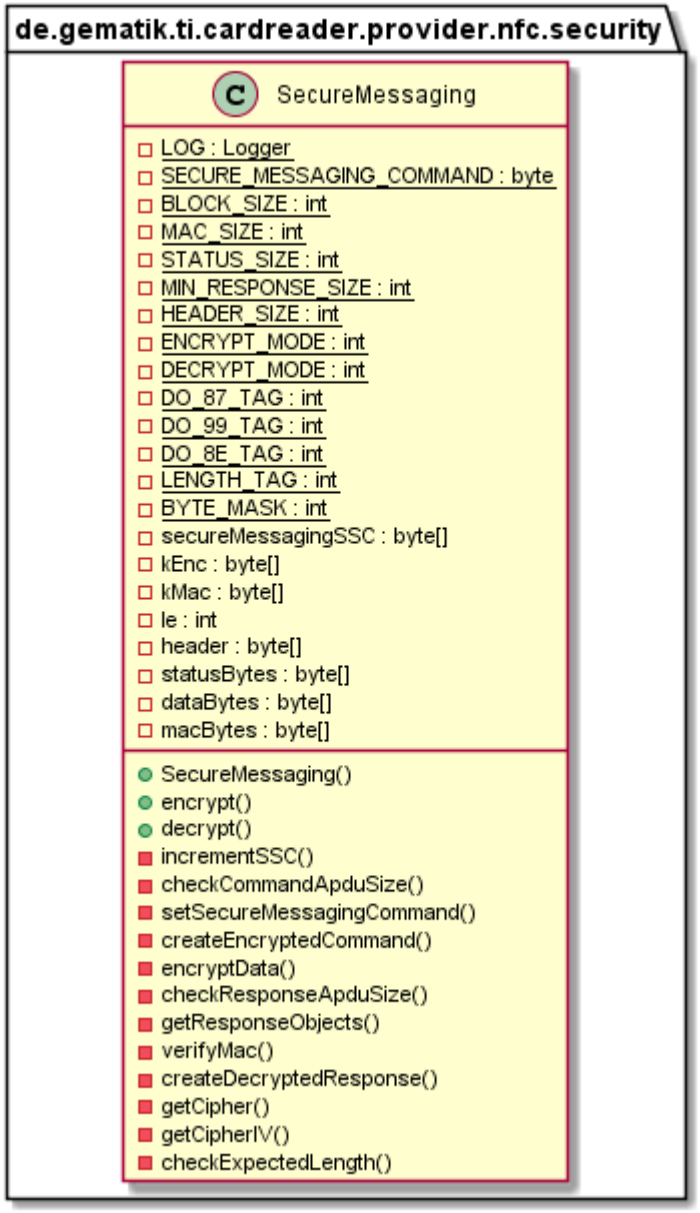
A logical channel connection to a Smart Card. It is used to exchange APDUs with a Smart Card the NfcCardReader class. A NfcCardChannel object can be obtained by calling the method nfcCard.getBasicChannel() or NfcCard.openLogicalChannel().




*Class diagram 43: NfcCardChannel*

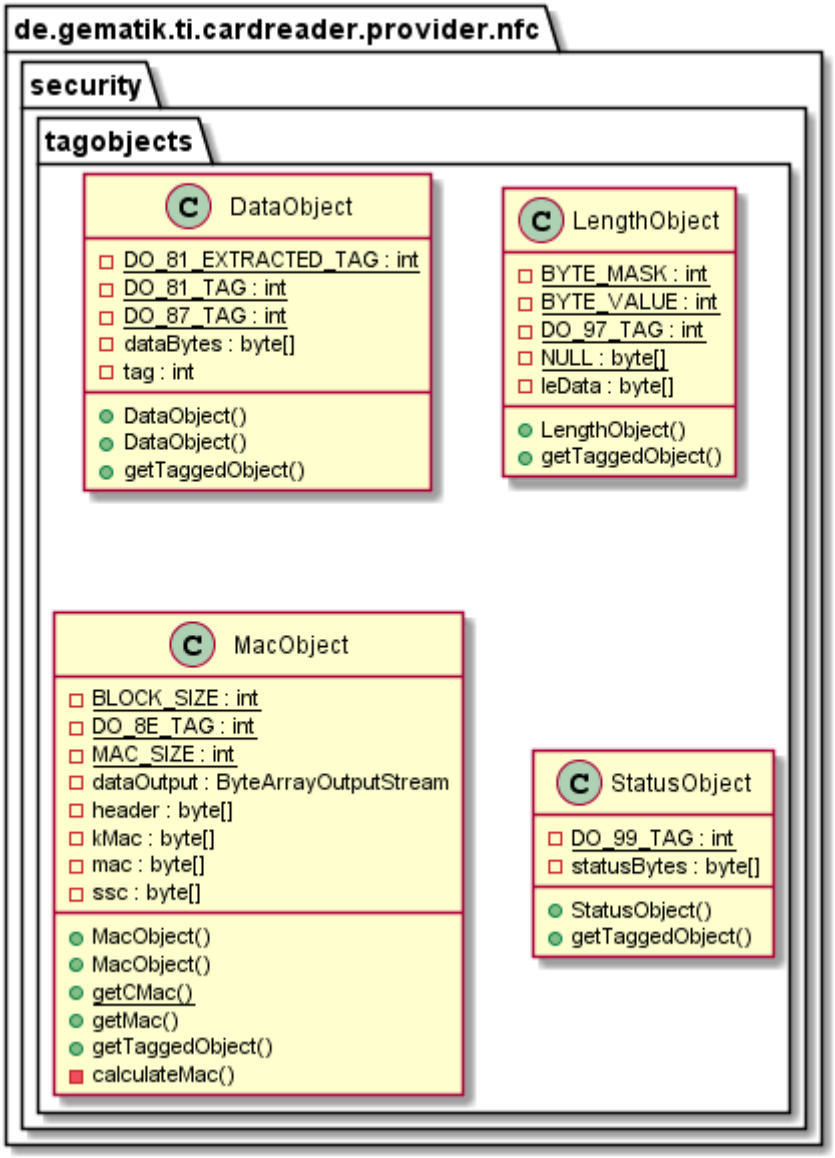
### 6.8.6. Security

Implements Secure Messaging for encrypting plain Command APDU and decrypting encrypted Response APDU.




*Class diagram 44: NfcSecureMessaging*

#### 6.8.6.1. TagObjects




*Class diagram 45: TagObjects*

## 6.8.7. Getting Started

### 6.8.7.1. Build setup

To use CardReaderProvider for NFC CardReader on Android-Device in a project, you need just to include following dependency:

*Listing 14. Gradle dependency settings to use CardReaderProvider for NFC CardReader on Android-Device library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'cardreader.provider.nfc', version:
'1.+'
}
```

*Listing 15. Maven dependency settings to use CardReaderProvider for NFC CardReader on Android-Device library*

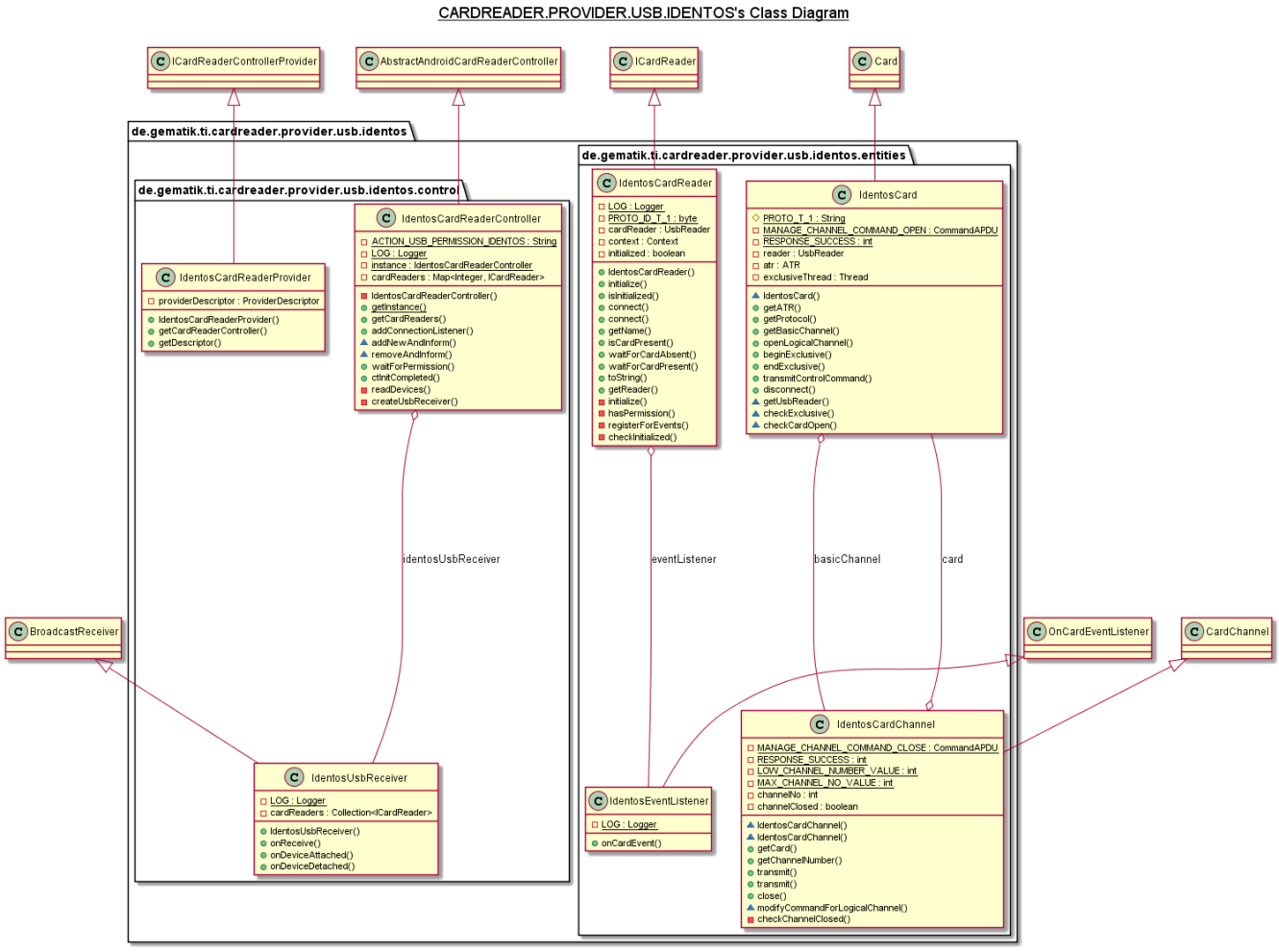
```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>cardreader.provider.nfc</artifactId>
        <version>1.+</version>
    </dependency>
</dependencies>
```

## 6.8.8. Identos USB CardReaderProvider

This part describes the usage of low level CardReaderProvider for USB IdentosCardReader in your application.

### 6.8.8.1. Structure

#### 6.8.8.1.1. Overview




*Class diagram 46: usbIdentosCardReaderProvider*

#### 6.8.8.1.2. Integration

The Identos USB CardReaderProvider needs a descriptor behind

`YOUR.PROVIDER\src\main\resources\META-INF\services` with filename `de.gematik.ti.cardreader.provider.spi.ICardReaderControllerProvider` and the content of the package and class which implements the service provider interface `de.gematik.ti.cardreader.provider.usb.identos.control.IdentosCardReaderProvider`.

#### 6.8.8.2. Hardware

The integrated Identos id100 library for Android supports the usage of integrating smart card functionality using Identos CardReader.

#### 6.8.8.3. About

##### 6.8.8.3.1. Licence

Gematik internal usage, details tbd.

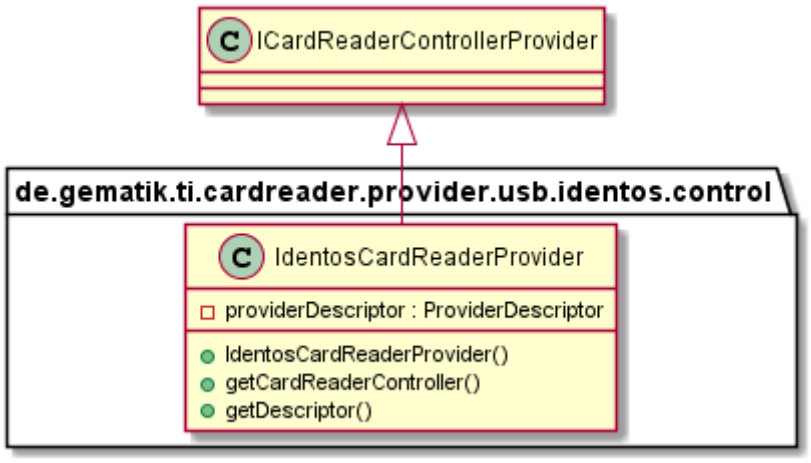
##### 6.8.8.3.2. Publisher

gematik Gesellschaft für Telematikanwendungen der Gesundheitskarte mbH 2019

#### 6.8.8.4. Control

##### 6.8.8.4.1. IdentosCardReaderProvider

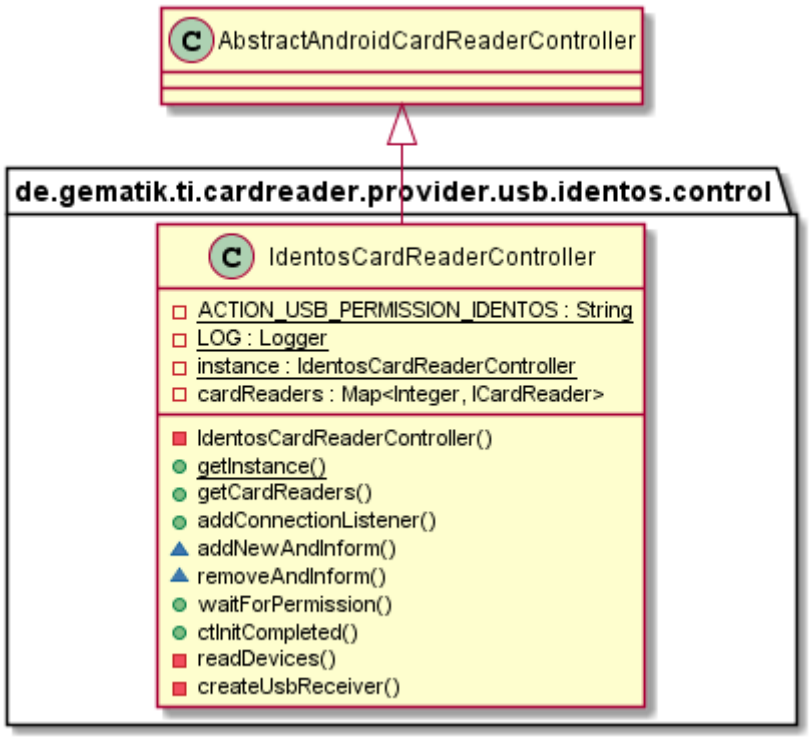
The IdentosCardReaderProvider class needs implementation of the interface 'ICardReaderControllerProvider' to handle listener and provide methods to inform connected listeners about card reader changes.




*Class diagram 47: IdentosCardReaderProvider*

##### 6.8.8.4.2. IdentosCardReaderController

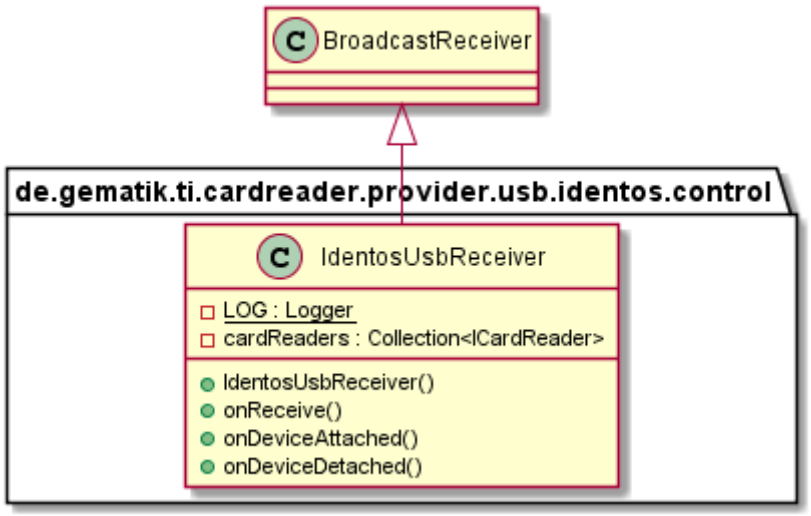
The IdentosCardReaderController class extends the abstract class 'AbstractCardReaderController' to handle necessary permissions and checking if the application context is set. Returns a list with currently connected Identos cardReaders and informs about reader connection and disconnection.




*Class diagram 48: IdentosCardReaderController*

#### 6.8.8.4.3. IdentosUsbReceiver

The IdentosUsbReceiver class extends the BroadCastReceiver class that receives and handles broadcast intents sent by {@link android.content.Context#sendBroadcast(Intent)}. The actions of the intents that are being handled are changes in the state of the usb device (attach / detach).

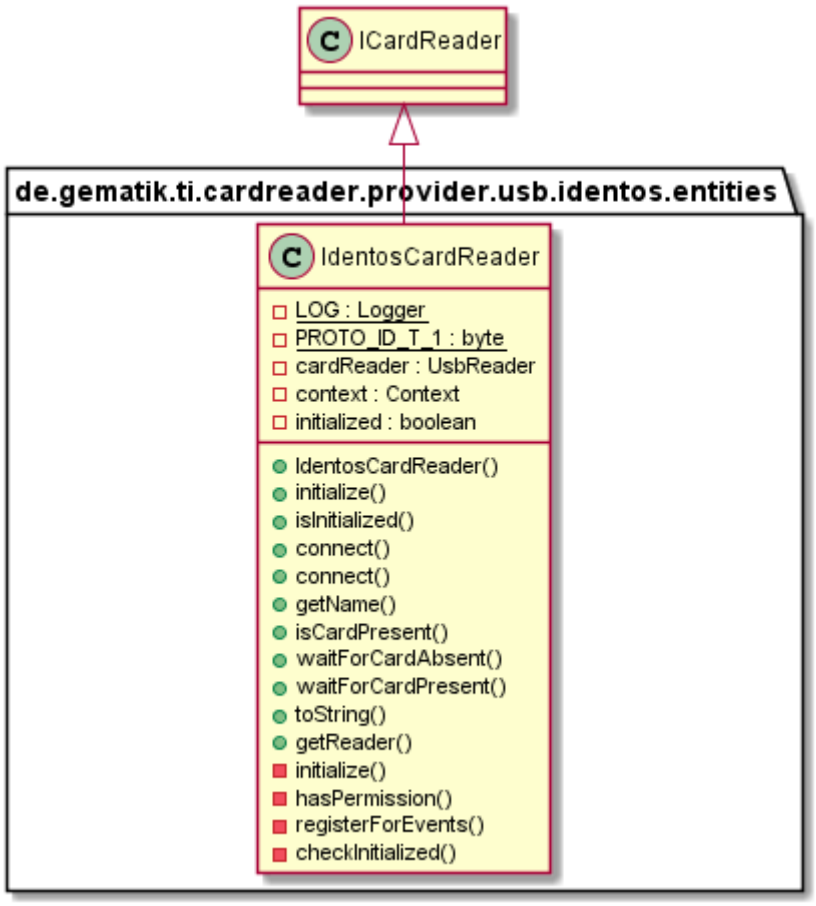



*Class diagram 49: IdentosUsbReceiver*

### 6.8.8.5. Entities

#### 6.8.8.5.1. IdentosCardReader

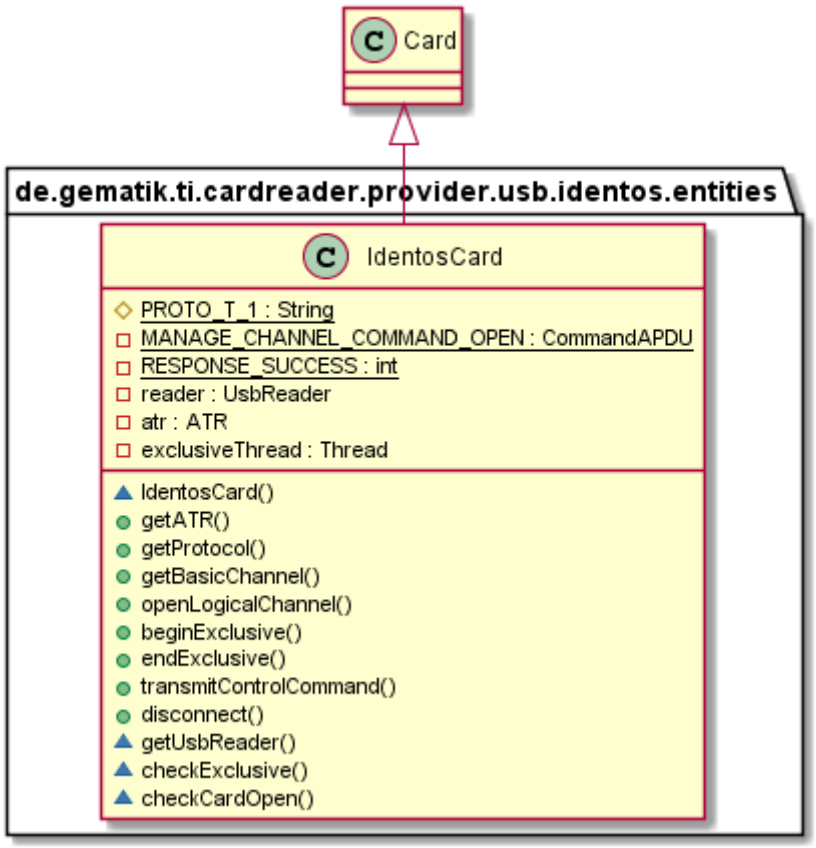
The Identos Smart Card Reader implements the Interface ICardReader. Works as adapter to the Identos id100 library for Android.




*Class diagram 50: IdentosCardReader*

#### 6.8.8.5.2. IdentosCard

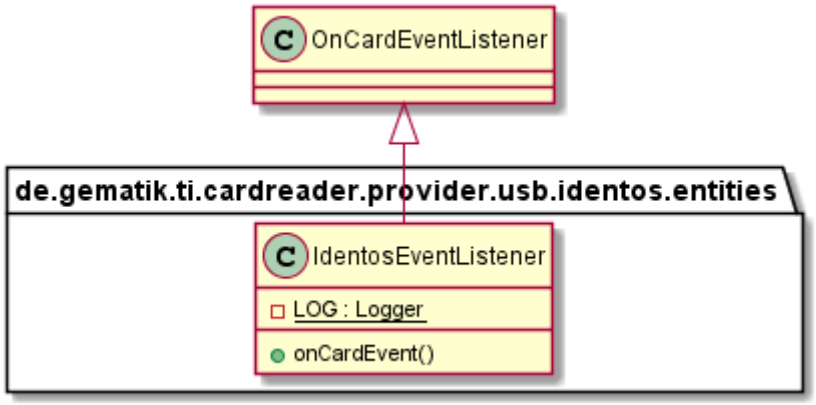
A Smart Card with which a connection has been established. IdentosCard extends the abstract class Card. Works as adapter to the Identos id100 library for Android.




*Class diagram 51: IdentosCard*

#### 6.8.8.5.3. IdentosEventListener

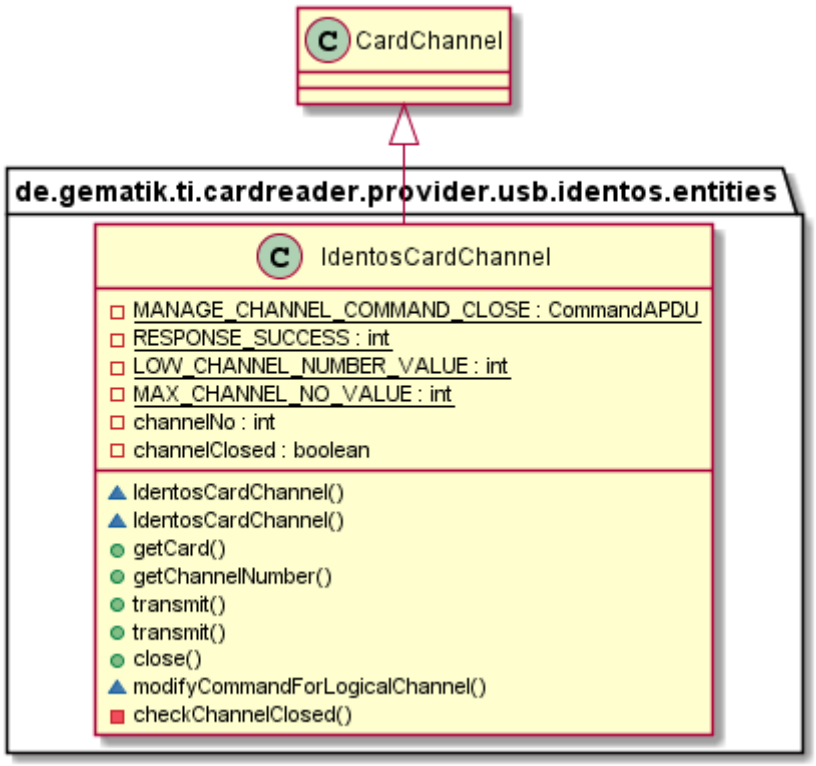
EventListener for card events. Works as adapter to the Identos id100 library for Android.




*Class diagram 52: IdentosEventListener*

#### 6.8.8.5.4. IdentosCardChannel

A logical channel connection to a Smart Card. It is used to exchange APDUs with a Smart Card using USB Identos CardReader. A IdentosCardChannel object can be obtained by calling the method IdentosCard.getBasicChannel() or IdentosCard.openLogicalChannel().




*Class diagram 53: IdentosCardChannel*

#### 6.8.8.6. Getting Started

##### 6.8.8.6.1. Build setup

To use CardReaderProvider for Identos USB CardReader in a project, you need just to include following dependency:

*Listing 16. Gradle dependency settings to use CardReaderProvider for Identos USB CardReader library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'cardreader.provider.usb.identos',
version: '1.+'
}
```

*Listing 17. Maven dependency settings to use CardReaderProvider for Identos USB library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>cardreader.provider.usb.identos</artifactId>
        <version>1.+</version>
    </dependency>
</dependencies>
```

### 6.8.9. Tactivo USB CardReaderProvider

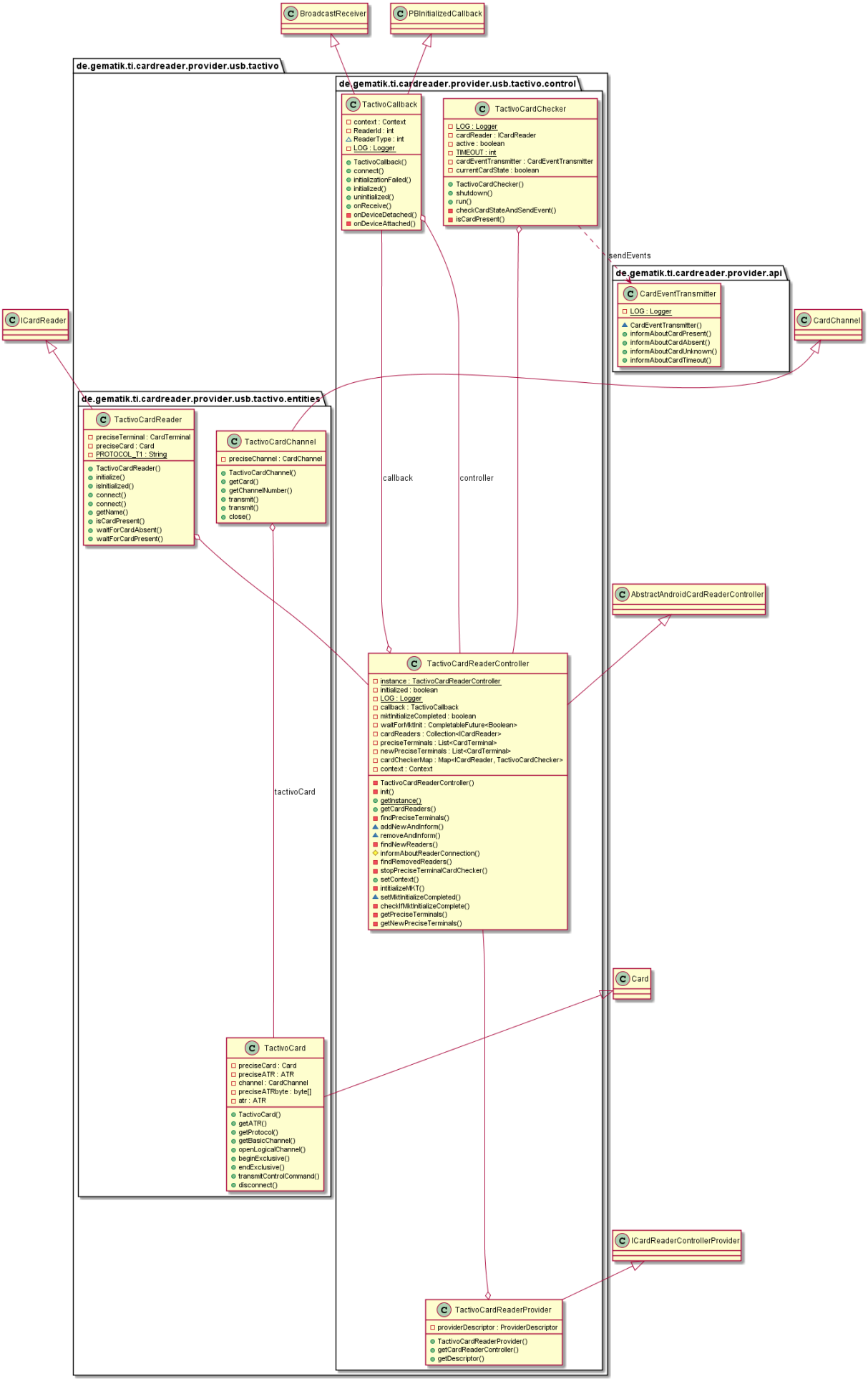
This part describes the usage of low level CardReaderProvider for USB Tactivo CardReader in your

application.

#### 6.8.9.1. Structure

##### 6.8.9.1.1. Overview

CARDREADER.PROVIDER.USB.TACTIVO's Class Diagram



*Class diagram 54: usbTactivoCardReaderProvider*

##### 6.8.9.1.2. Integration

The Tactivo USB CardReaderProvider needs a descriptor behind `YOUR.PROVIDER\src\main\resources\META-INF\services` with filename `de.gematik.ti.cardreader.provider.spi.ICardReaderControllerProvider` and the content of the package and class which implements the service provider interface `de.gematik.ti.cardreader.provider.usb.tactivo.control.TactivoCardReaderProvider`.

### 6.8.9.2. Hardware

The integrated Precise Mobile Toolkit for Android SDK supports the integrating of smart card and fingerprint functionality using Tactivo mini.

### 6.8.9.3. Additional Software

To initialize a Tactivo cardReader, it is necessary to install the Tactivo Manager App on the device. (https://play.google.com/store/apps/details?id=com.precisebiometrics.android.mtk.manager&hl=gsw)

### 6.8.9.4. About

#### 6.8.9.4.1. Licence

Gematik internal usage, details tbd.

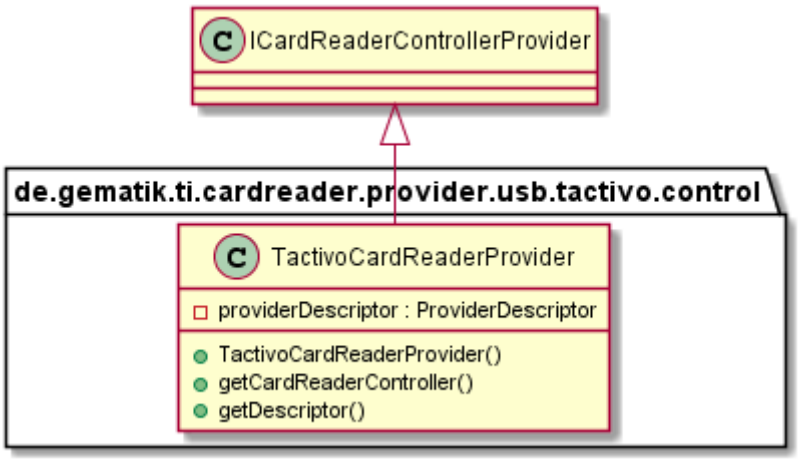
#### 6.8.9.4.2. Publisher

gematik Gesellschaft für Telematikanwendungen der Gesundheitskarte mbH 2019

### 6.8.9.5. Control

#### 6.8.9.5.1. TactivoCardReaderProvider

The TactivoCardReaderProvider class needs implementation of the interface 'ICardReaderControllerProvider' to handle listener and provide methods to inform connected listeners about card reader changes.




*Class diagram 55: TactivoCardReaderProvider*

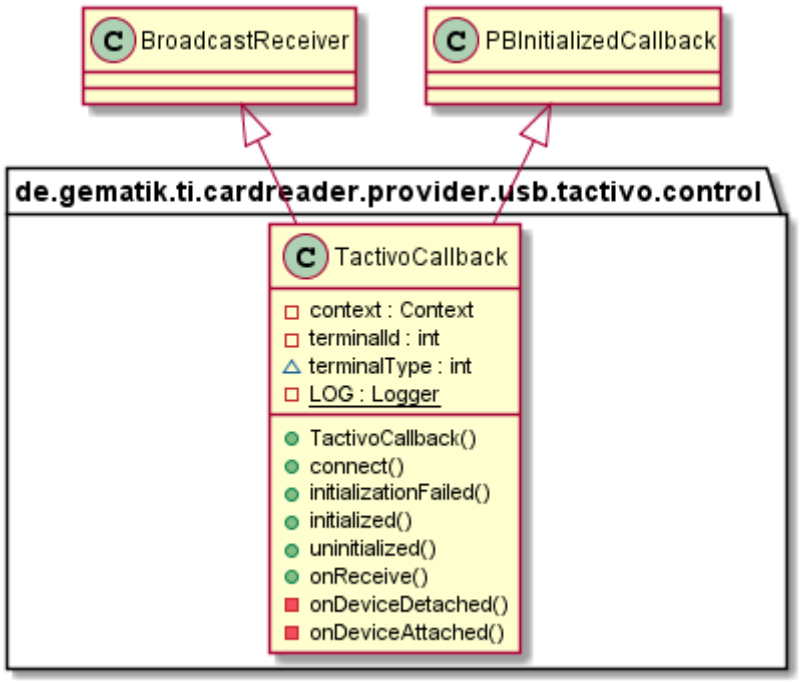
#### 6.8.9.5.2. TactivoCardReaderController

The TactivoCardReaderController extends the abstract class 'AbstractCardReaderController' to handle necessary permissions and checking if the application context is set. Returns a list with currently connected Tactivo cardReaders and informs about reader connection and disconnection. Additionally start and stop the controller for each TactivoCardReader the associated TactivoCardChecker.


*Class diagram 56: TactivoCardReaderController*

#### 6.8.9.5.3. TactivoCallback

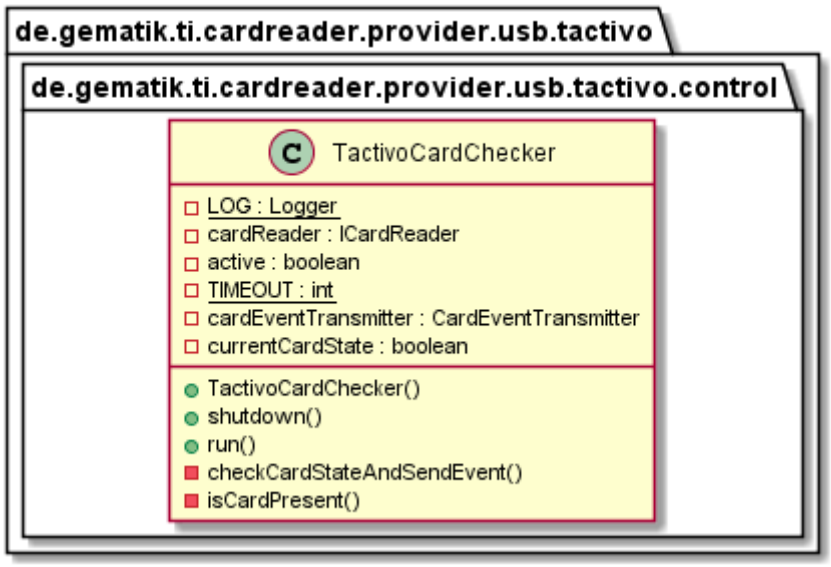
The TactivoCallback class implements the PBInitializedCallback class from Precise Mobile Toolkit for Android SDK to connect and initialize Tactivo CardReader and extends the BroadCastReceiver class that receives and handles broadcast intents sent by {@link android.content.Context#sendBroadcast(Intent)}. The actions of the intents that are being handled are changes in the state of the usb device (attach / detach).




*Class diagram 57: TactivoCallback*

#### 6.8.9.5.4. TactivoCardChecker

The TactivoCardChecker would automatically started for each connected TactivoCardReader to monitor the current card status. This checker send Events on EventBus for each present or absent card. For triggering this changed would use the SmartCardIo methods `waitForCardAbsent` and `waitForCardPresent`.

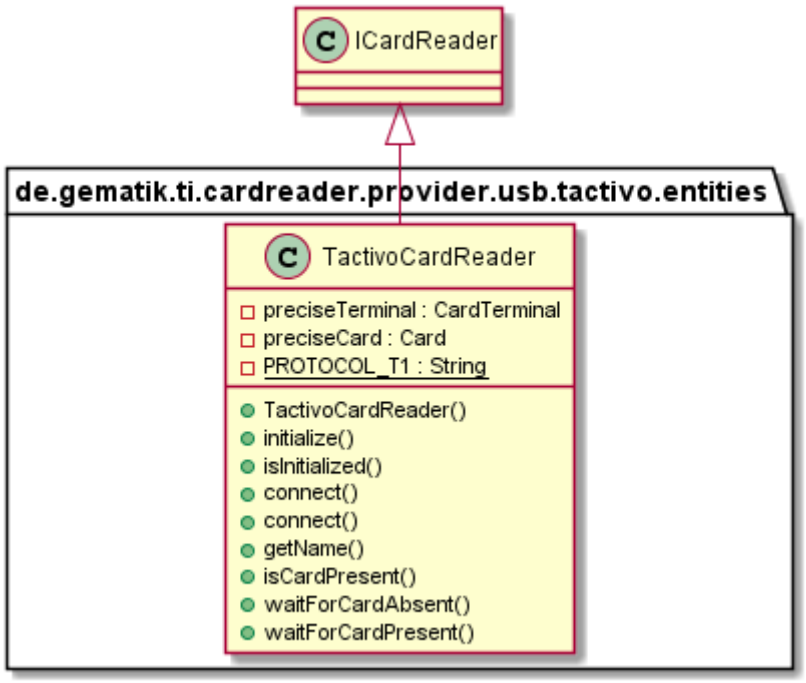



*Class diagram 58: TactivoCardChecker*

### 6.8.9.6. Entities

#### 6.8.9.6.1. TactivoCardReader

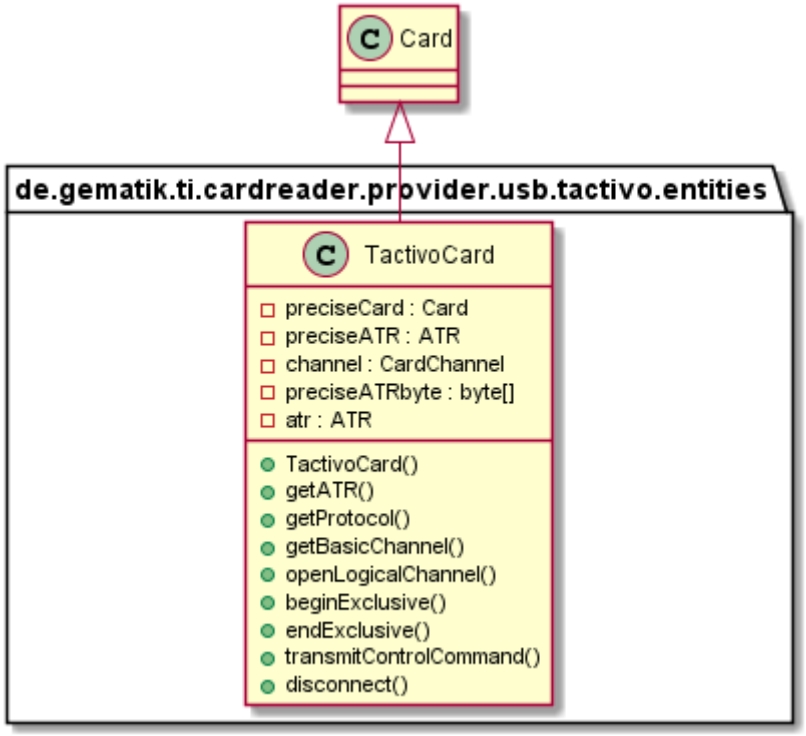
The Tactivo Smart Card Reader implements the Interface ICardReader. Works as adapter to the Precise Mobile Toolkit for Android SDK.




*Class diagram 59: TactivoCardReader*

#### 6.8.9.6.2. TactivoCard

A Smart Card with which a connection has been established. TactivoCard extends the abstract class Card. Works as adapter to the Precise Mobile Toolkit for Android SDK.

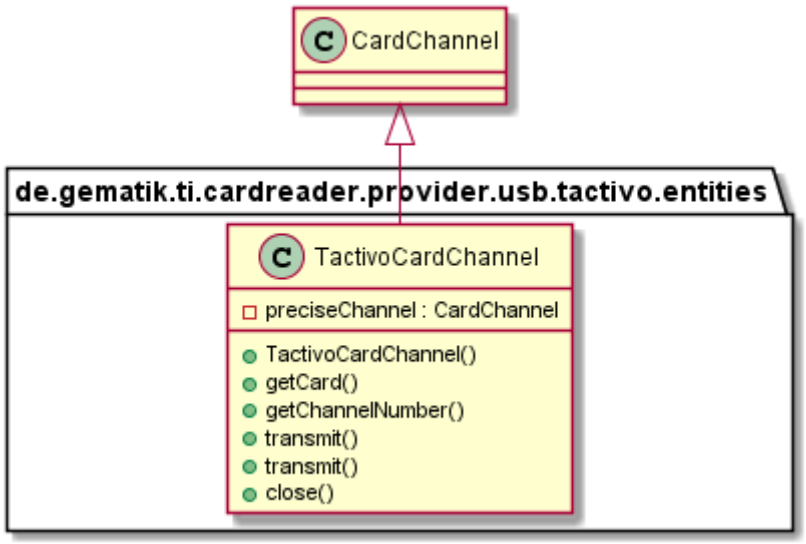



*Class diagram 60: TactivoCard*

#### 6.8.9.6.3. TactivoCardChannel

A logical channel connection to a Smart Card. It is used to exchange APDUs with a Smart Card using USB Tactivo CardReader. A TactivoCardChannel object can be obtained by calling the method TactivoCard.getBasicChannel() or TactivoCard.openLogicalChannel(). Works as adapter to the

Precise Mobile Toolkit for Android SDK.




*Class diagram 61: TactivoCardChannel*

#### 6.8.9.7. Getting Started

##### 6.8.9.7.1. Build setup

To use CardReaderProvider for Tactivo USB CardReader in a project, you need just to include following dependency:

*Listing 18. Gradle dependency settings to use CardReaderProvider for Tactivo USB CardReader library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'cardreader.provider.usb.tactivo',
version: '1.+'
}
```

*Listing 19. Maven dependency settings to use CardReaderProvider for Tactivo USB library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>cardreader.provider.usb.tactivo</artifactId>
        <version>1.+</version>
    </dependency>
</dependencies>
```

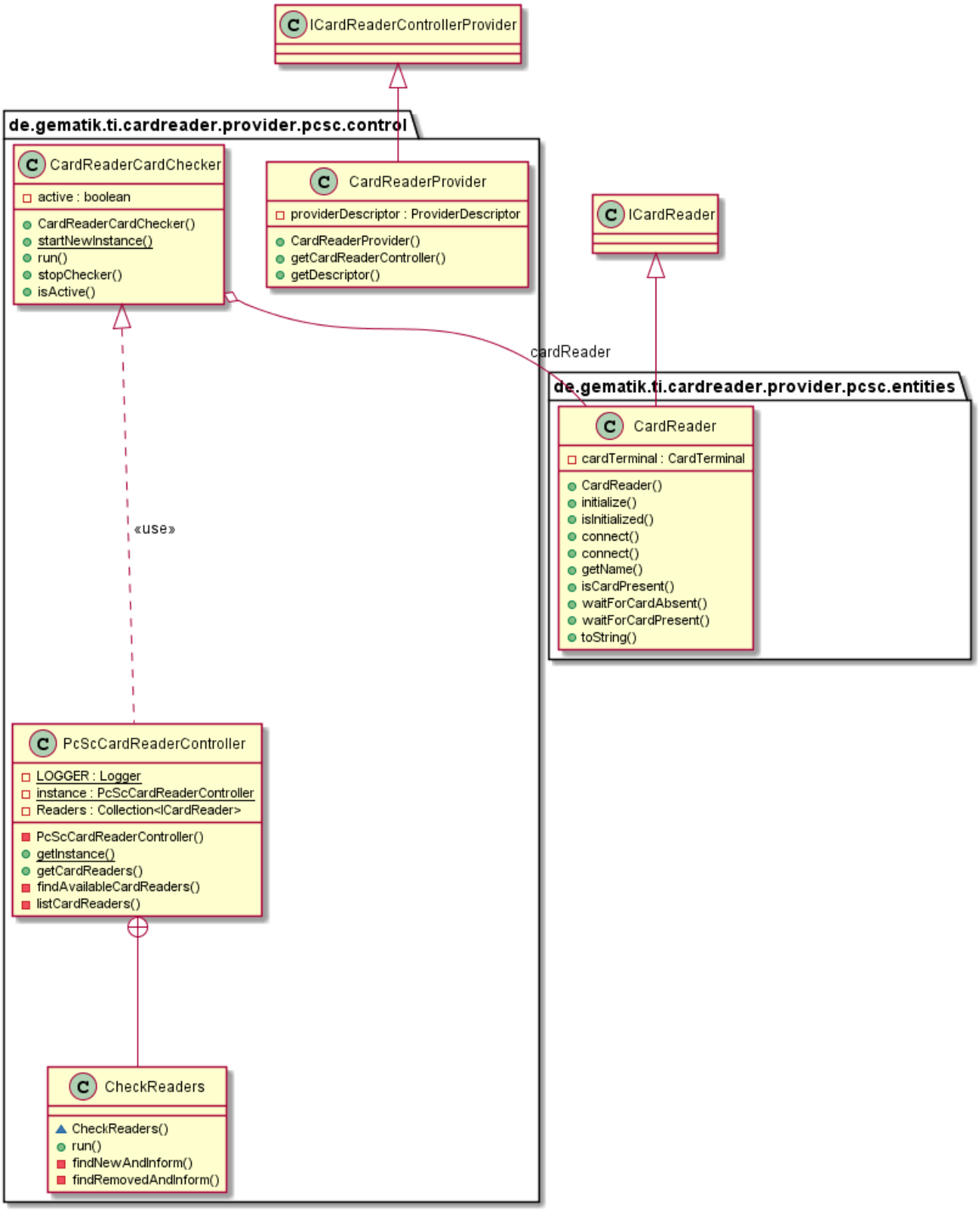
**To initialize a Tactivo cardReader, it is necessary to install the Tactivo Manager App on the device. (https://play.google.com/store/apps/details?id=com.precisebiometrics.android.mtk.manager&hl=gsw)**

## 6.8.10. PCSC CardReaderProvider

## 6.8.11. Introduction

This part describes the usage of low level CardReaderProvider for PCSC CardReader in your application.

## 6.8.12. Overview




*Class diagram 62: PcscCardReaderProvider*

#### 6.8.12.1. Integration

The PCSC CardReaderProvider needs a descriptor behind `YOUR.PROVIDER\src\main\resources\META-INF\services` with filename `de.gematik.ti.cardreader.provider.spi.ICardReaderControllerProvider` and the content of the package and class which implements the service provider interface `dde.gematik.ti.cardreader.provider.pcsc.control.CardReaderProvider`.

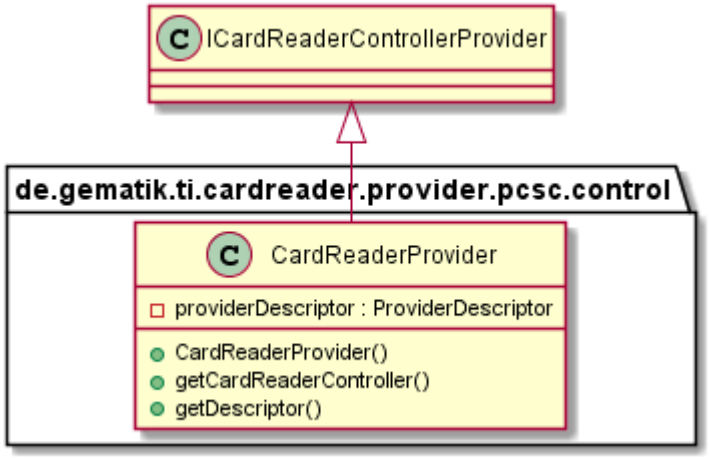
### 6.8.13. Hardware

Each card reader device which communicates over pcsc protocol

### 6.8.14. Control

#### 6.8.14.1. PCSCCardReaderProvider

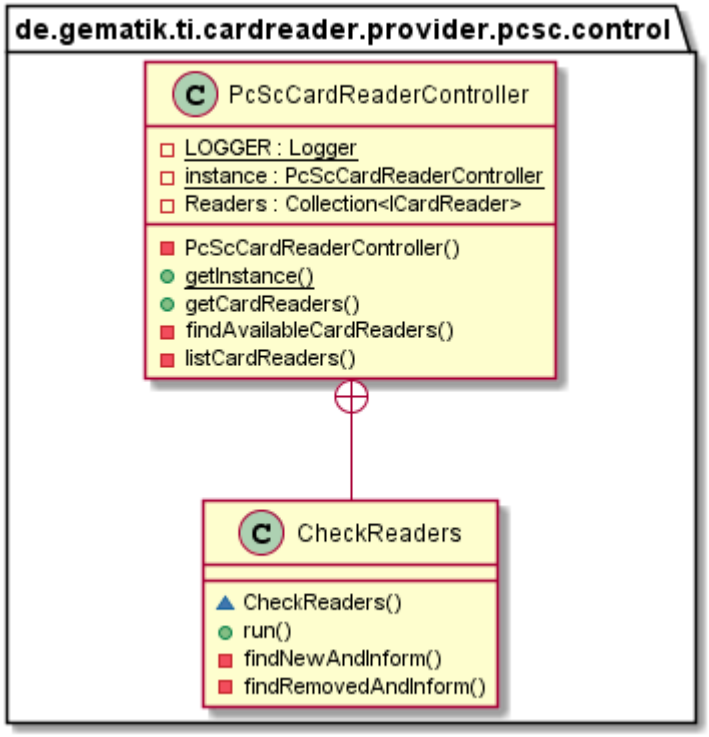
The PCSCCardReaderProvider class needs implementation of the interface 'ICardReaderControllerProvider' to handle listener and provide methods to inform connected listeners about card reader changes.




*Class diagram 63: PCSCCardReaderProvider*

#### 6.8.14.2. PCSCCardReaderController

The PCSCCardReaderController class extends the abstract class 'AbstractCardReaderController' to handle necessary permissions and checking if the application context is set. Returns a list with currently connected pcsc cardReaders and informs about reader connection and disconnection.

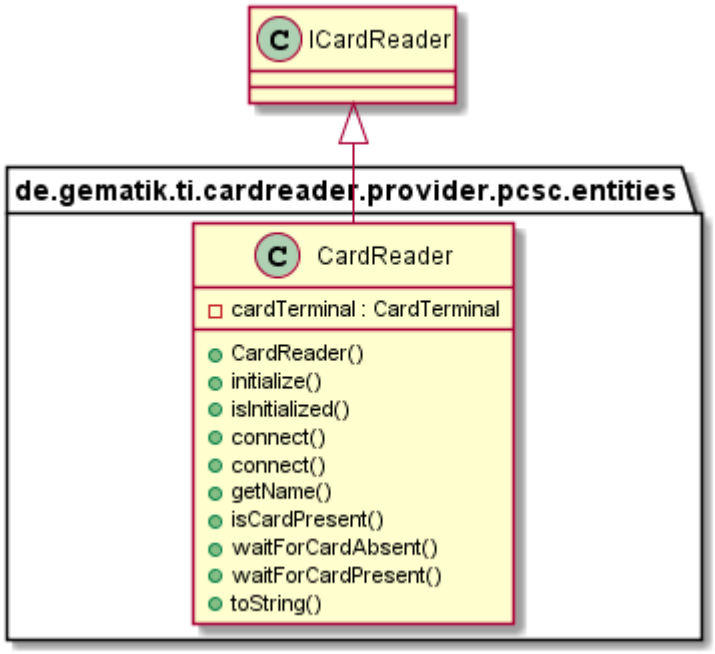



*Class diagram 64: PCSCCardReaderController*

## 6.8.15. Entities

### 6.8.15.1. CardReader

The CardReader represent a pcsc card reader with one slot.




*Class diagram 65: CardReader*

## 6.8.16. Getting Started

#### 6.8.16.1. Build setup

To use CardReaderProvider for Tactivo USB CardReader in a project, you need just to include following dependency:

*Listing 20. Gradle dependency settings to use PCSC CardReaderProvider library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'cardreader.provider.pcsc', version:
'1.1.2'
}
```

*Listing 21. Maven dependency settings to use PCSC CardReaderProvider library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>cardreader.provider.pcsc</artifactId>
        <version>1.1.2</version>
    </dependency>
</dependencies>
```

# Chapter 7. Ti-Utils

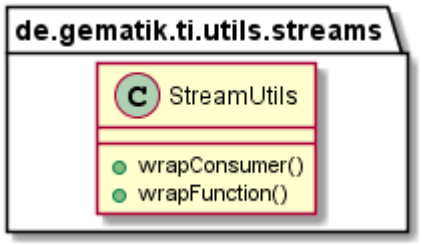
## 7.1. Introduction

This part describes the Ti-Utils functionalities and structure.

## 7.2. Overview

### 7.2.1. StreamUtils

The StreamUtils functions capture exception thrown by execution on streams. With special funktions will wrap the call on orginal object and all exception wrapped as `StreamRuntimeException`.

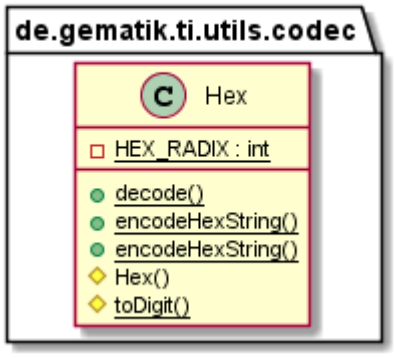



*Class diagram 66: StreamUtils structure*

The usage is described in GettingStarted Ti-Utils

### 7.2.2. Codec

This package contains a Hex class with converter functionality for hex strings. The Hex class is inspired from org.apache.commons.codec.binary.Hex (Licence: http://www.apache.org/licenses/). Detailed information will found in javadoc of this project.

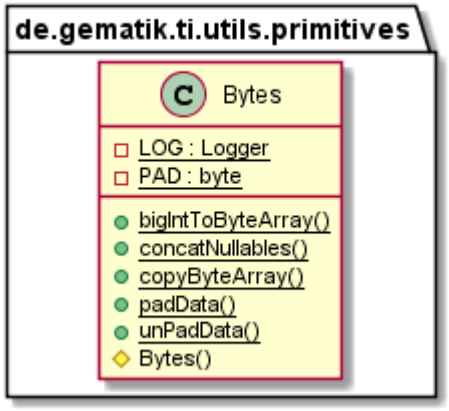



*Class diagram 67: Codec package structure*

### 7.2.3. Primitives

This package contains a Bytes class with utils functionality for byte arrays. The Hex class is inspired from com.google.common.primitives.Bytes (Apache License 2.0) http://www.apache.org/licenses/LICENSE-2.0.html See the Guava User Guide article on primitive utilities.

Detailed information will found in javadoc of this project.




*Class diagram 68: Primitives package structure*

# 7.3. Getting Started

## 7.3.1. Build setup

To use Ti-Utils library in a project, you need just to include following dependency:

*Listing 22. Gradle dependency settings to use Ti-Utils library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'utils', version: '{version_TIUTILS}'
}
```

*Listing 23. Maven dependency settings to use Ti-Utils library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>utils</artifactId>
        <version>{version_TIUTILS}</version>
    </dependency>
</dependencies>
```

## 7.3.2. StreamUtils

The StreamUtils methods are available as static call with `de.gematik.ti.utils.streams.stat.StreamUtils` and as instance call from `de.gematik.ti.utils.streams.StreamUtils`.

### 7.3.2.1. WrapConsumer call

*Listing 24. WrapConsumer call with StreamUtils*

```
private List<ExceptionClass> list;
list = Arrays.asList(new ExceptionClass(), new ExceptionClass(), new
ExceptionClass());

list.stream().forEach(new StreamUtils().wrapConsumer(v -> v.testMethod()));
```

This is also available as a static method call with `StreamUtils` from package `de.gematik.ti.utils.streams.stat`

*Listing 25. WrapConsumer call with static StreamUtils*

```
private List<ExceptionClass> list;
list = Arrays.asList(new ExceptionClass(), new ExceptionClass(), new
ExceptionClass());

list.stream().forEach(StreamUtils.wrapConsumer(v -> v.testMethod()));
```

#### 7.3.2.2. WrapFunction call

*Listing 26. WrapFunction call with StreamUtils*

```
private List<ExceptionClass> list;
list = Arrays.asList(new ExceptionClass(), new ExceptionClass(), new
ExceptionClass());

list.stream().map(new StreamUtils().wrapFunction(v ->
v.getValue())).collect(Collectors.toList());
```

This is also available as a static method call with `StreamUtils` from package `de.gematik.ti.utils.streams.stat`

*Listing 27. WrapFunction call with static StreamUtils*

```
private List<ExceptionClass> list;
list = Arrays.asList(new ExceptionClass(), new ExceptionClass(), new
ExceptionClass());

list.stream().map(StreamUtils.wrapFunction(v ->
v.getValue())).collect(Collectors.toList());
```

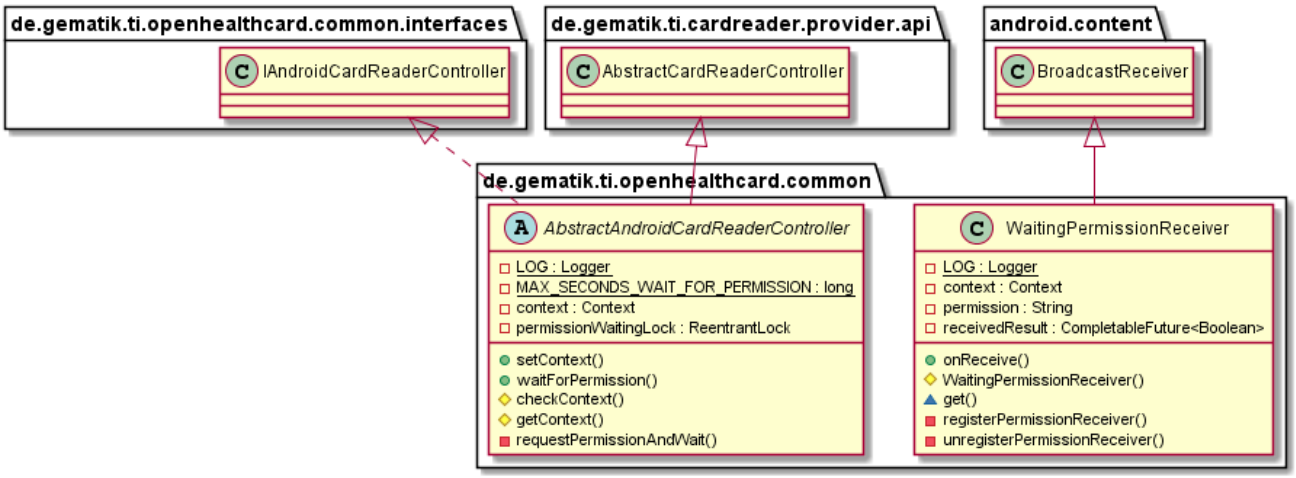
# Chapter 8. OpenHealthCard-Common Android Library

## 8.1. Introduction

This part describes the functionality and usability of android common library

## 8.2. Overview

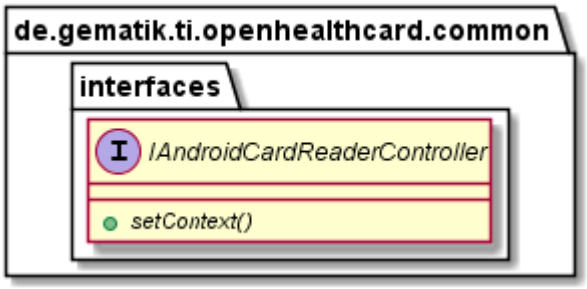
This Library contains Controller to inject Android Context to CardReader Provider. For that extends this Library the cardreader.provider.api functionality for Android. Furthermore Controller to request Permissions from Provider to handle e.g. card reader.



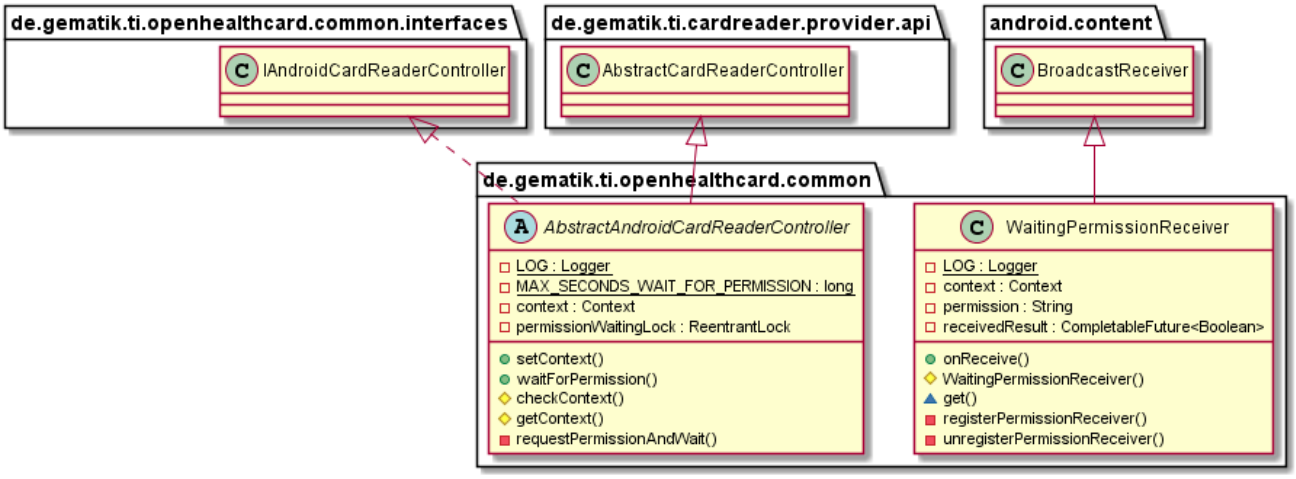

*Class diagram 69: OpenHealthCard Common Android library overview*

### 8.2.1. AndroidCardReaderController

This class is an android specific extension of AbstractCardReaderController from package de.gematik.ti.cardreader.provider.api. The additional functionalities are android context handling and permissions requests, necessary for android card reader provider.




*Class diagram 70: OpenHealthCard Common Android library overview*




*Class diagram 71: OpenHealthCard AbstractCardReaderController*

# 8.3. Getting Started

## 8.3.1. Build setup

To use OpenHealthCard Common library in a project, you need just to include following dependency:

*Listing 28. Gradle dependency settings to use OpenHealthCard Common library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'openhealthcard.common', version:
'{version_OHCCOM}'
}
```

*Listing 29. Maven dependency settings to use OpenHealthCard Common library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>openhealthcard.common</artifactId>
        <version>{version_OHCCOM}</version>
    </dependency>
</dependencies>
```

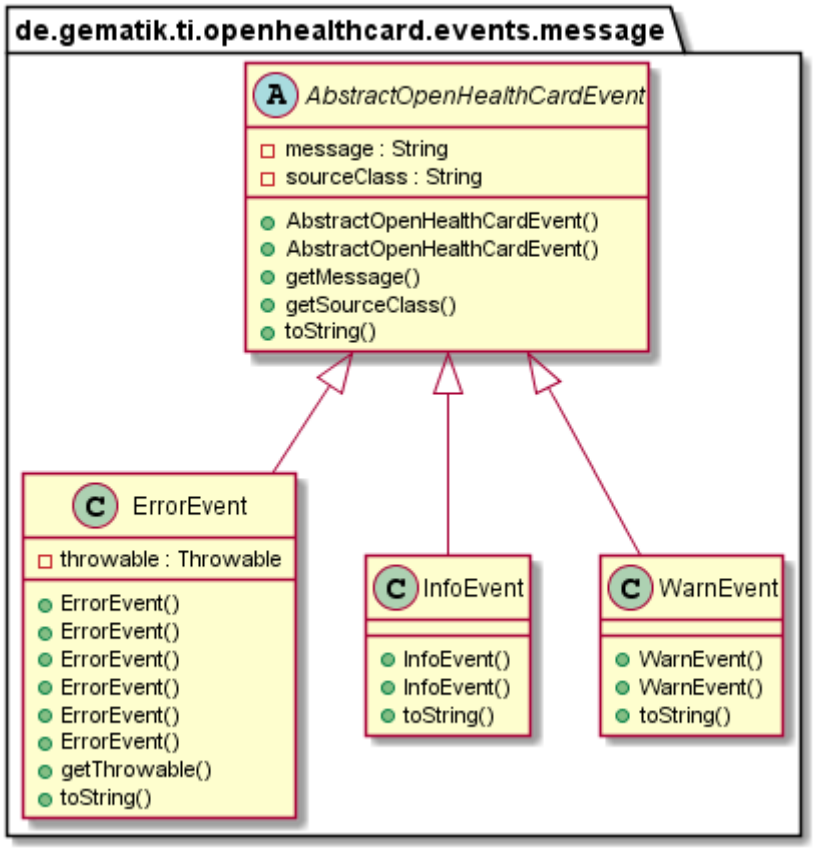
# 8.4. OpenHealthCard Events

# 8.5. Introduction

This part describes the OpenHealthCard-Events functionalities and structure.

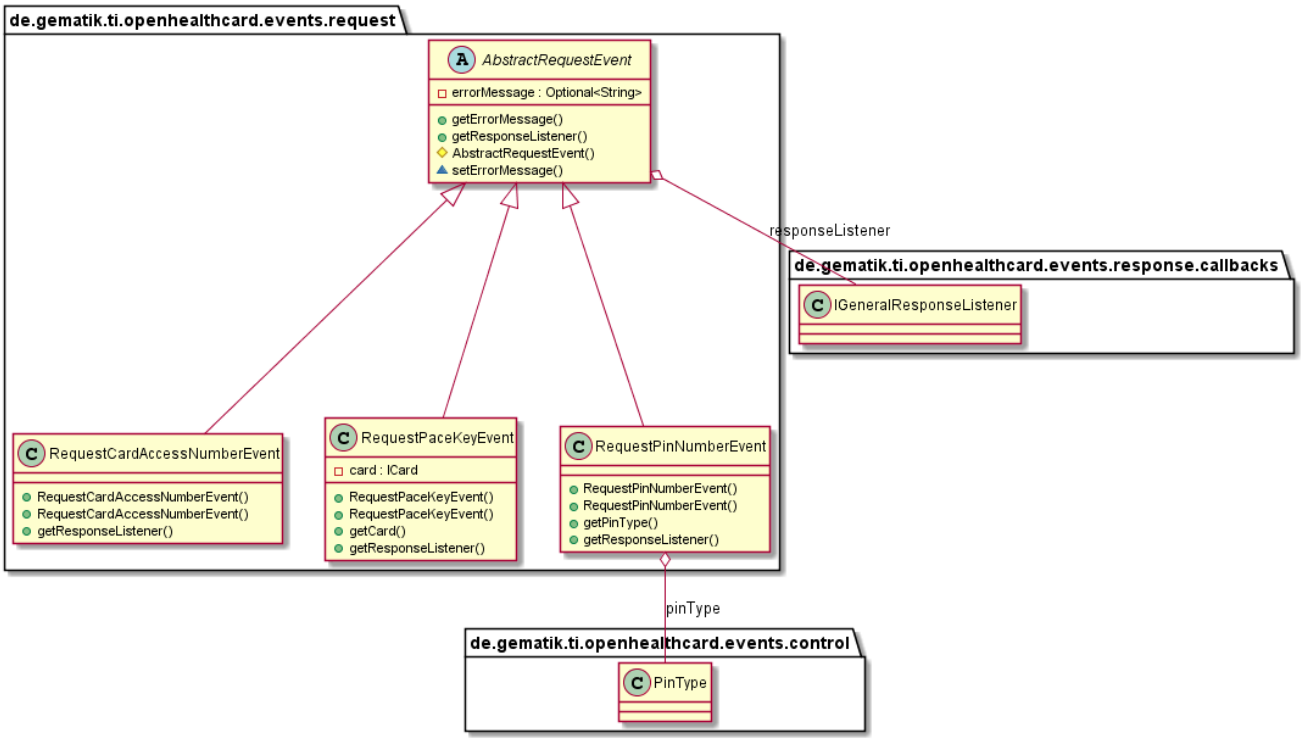
## 8.6. Overview

This library contains overarching events for several layers. That includes general events for info, warn and error



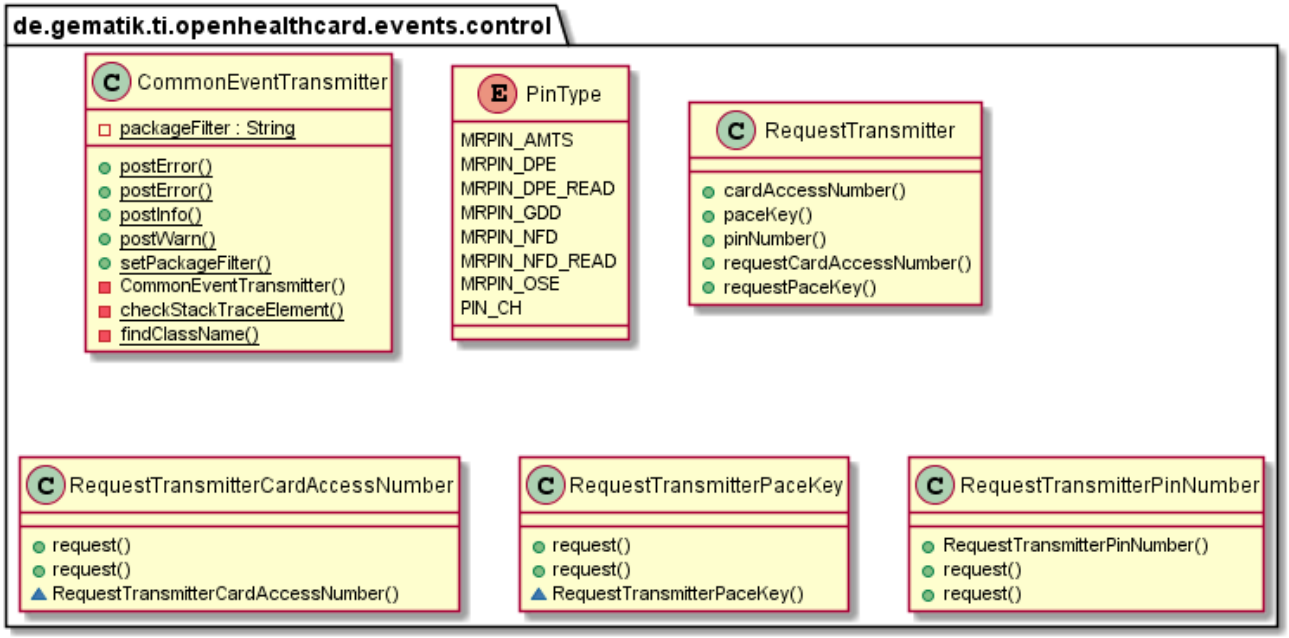

*Class diagram 72: OpenHealthCard-Events general Messages*

Additionally request events to get asynchronous input from user application




*Class diagram 73: OpenHealthCard-Events request events*

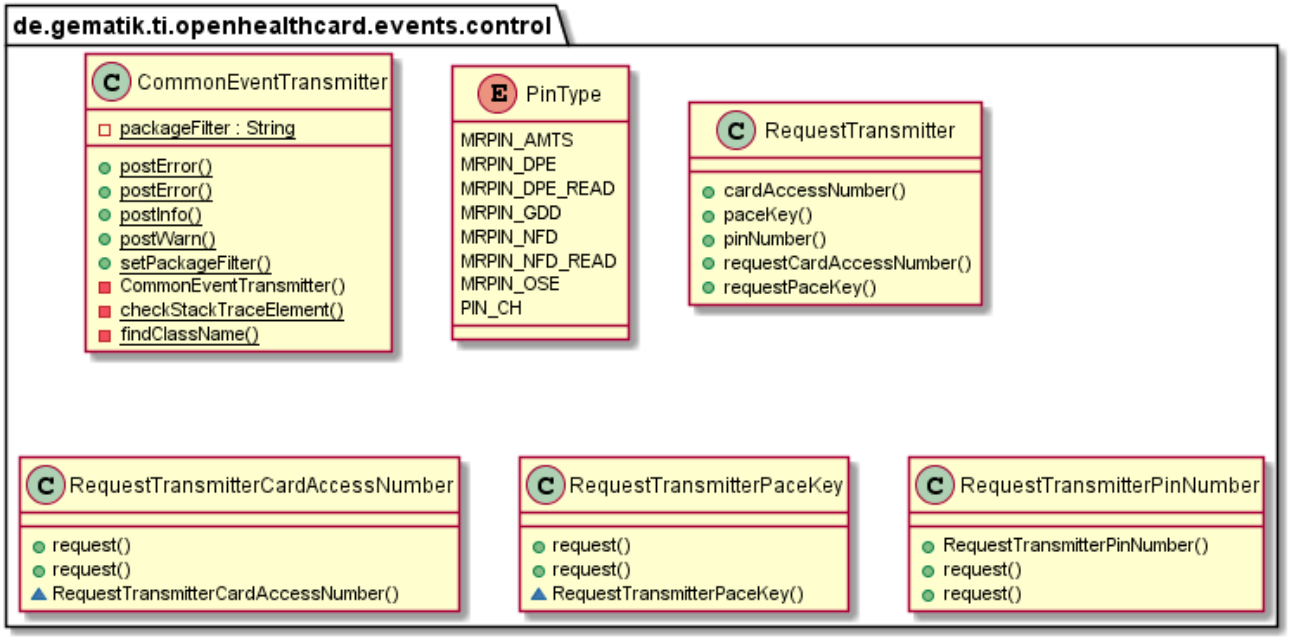
For easy use could you use the controller instances to send events to EventBus.




*Class diagram 74: OpenHealthCard-Events event controller*

### 8.6.1. EventTransmitter

For easy use could you use the controller instances to send events to EventBus. This Controller Instances are `CommonEventTransmitter` to send information, `RequestTransmitter` to generate the context specific Transmitter like PaceKey, PinNumber, etc.




*Class diagram 75: OpenHealthCard-Events event controller*

#### 8.6.1.1. CommonEventTransmitter

For easy usage could you send information, warning or error event with a simple method call like

*Listing 30. Send Exception as Event*

```
CommonEventTransmitter.postError(someException, myMessage);
```

Receive an Information Event like:

*Listing 31. Receive Infromation-Event*

```
@Subscribe(threadMode = ThreadMode.MAIN)
public void handleMessagesFromEventBus(final AbstractOpenHealthCardEvent event) {
    if (event instanceof ErrorEvent) {
        LOG.error(event.getSourceClass() + ": " + event.getMessage());
    } else {
        LOG.info(event.getSourceClass() + ": " + event.getMessage());
    }
    Toast.makeText(this, event.getSourceClass() + ": " + event.getMessage(),
Toast.LENGTH_LONG).show();
}
```

#### 8.6.1.2. RequestTransmitter

Factory to create the context specific Transmitter.

#### 8.6.1.3. RequestTransmitter for CardAccessNumber

This Transmitter create and send the CardAccessNumber specific request events

*Listing 32. Send CardAccessNumber request*

```
ICardAccessNumberResponseListener cardAccessNumberResponseListener = new
ICardAccessNumberResponseListener() {
    @Override
    public void handleCan(final CardAccessNumber cardAccessNumber) {
        // Response if the User Input is finished
    }
};

new RequestTransmitter().cardAccessNumber().request(cardAccessNumberResponseListener);
```

If you would inform receivers about failures on previous card access number requests could you send an message as second parameter:

*Listing 33. Send CardAccessNumber request with message*

```
new RequestTransmitter().cardAccessNumber().request(cardAccessNumberResponseListener,
someMessageToUser);
```

Receive an CardAccessNumber event and response to the requester

*Listing 34. Send CardAccessNumber request*

```
@Subscribe(threadMode = ThreadMode.MAIN)
public void requestCan(final RequestCardAccessNumberEvent event) {
    CardAccessNumber can = null;
    // Do some user interaction to get the card access number
    event.getResponseListener().handleCan(can);
}
```

#### 8.6.1.4. RequestTransmitter for PaceKey

This Transmitter create and send the PaceKey specific request events

*Listing 35. Send PaceKey request*

```
IPaceKeyResponseListener paceKeyResponseListener = new IPaceKeyResponseListener() {
    @Override
    public void handlePaceKey(final PaceKey paceKey) {
        // Do something with paceKey
    }
};

new RequestTransmitter().paceKey().request(paceKeyResponseListener,
cardFromCardReader);
```

If you would inform receivers about failures on previous pace key requests could you send an message as second parameter:

*Listing 36. Send PaceKey request with message*

```
new RequestTransmitter().paceKey().request(paceKeyResponseListener,
cardFromCardReader, someMessageToUser);
```

Receive an PaceKey event and response to the requester

*Listing 37. Send PaceKey request*

```
@Subscribe(threadMode = ThreadMode.MAIN)
public void requestPaceKey(final RequestPaceKeyEvent event) {
    PaceKey paceKey = null;
    // Do some user interaction to get the card access number
    event.getResponseListener().handlePaceKey(paceKey);
}
```

#### 8.6.1.5. RequestTransmitter for PinNumber

This Transmitter create and send the PinNumber specific request events

*Listing 38. Send PinNumber request*

```
IPinNumberResponseListener pinNumberResponseListener = new
IPinNumberResponseListener() {
    @Override
    public void handlePinNumber(final PinNumber pinNumber) {
        // Response if the User Input is finished
    }
};

new RequestTransmitter().pinNumber().request(pinNumberResponseListener, pinType);
```

If you would inform receivers about failures on previous pin number requests could you send an message as second parameter:

*Listing 39. Send PinNumber request with message*

```
new RequestTransmitter().pinNumber().request(pinNumberResponseListener, pinType,
someMessageToUser);
```

Receive an PinNumber event and response to the requester

*Listing 40. Send PinNumber request*

```
@Subscribe(threadMode = ThreadMode.MAIN)
public void requestPin(final RequestPinNumberEvent event) {
    PinNumber pinNumber = null;
    // Do some user interaction to get the card access number
    event.getResponseListener().handlePinNumber(pinNumber);
}
```

# 8.7. Getting Started

## 8.7.1. Build setup

To use OpenHealthCard-Event library in a project, you need just to include following dependency:

*Listing 41. Gradle dependency settings to use OpenHealthCard-Event library*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'openhealthcard.events', version:
'1.+'
}
```

*Listing 42. Maven dependency settings to use OpenHealthCard-Events library*

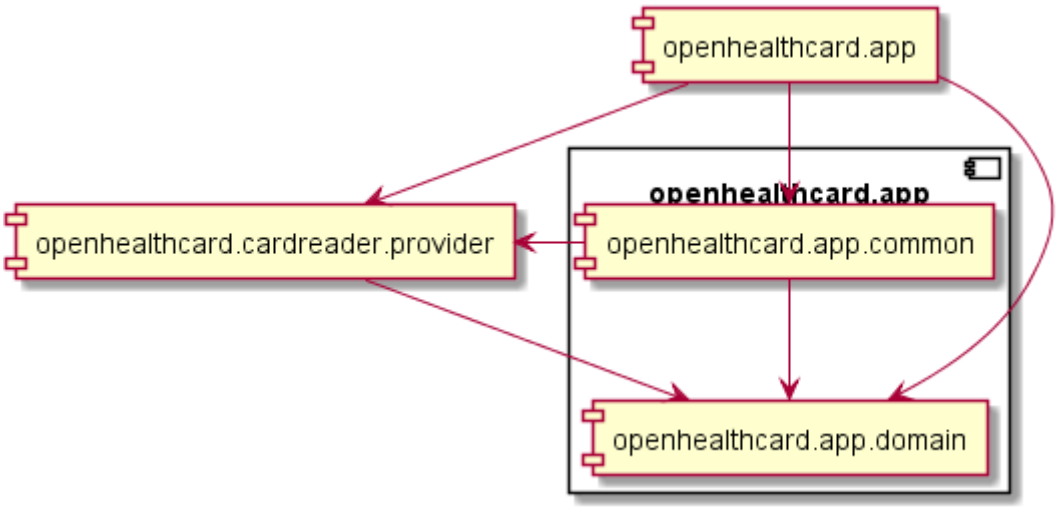
```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>openhealthcard.events</artifactId>
        <version>1.+</version>
    </dependency>
</dependencies>
```

# 8.8. Open Health Card Android Application

This part describes the structure and functionality of OpenHealthCard App for Android

## 8.8.1. Structure

#### 8.8.1.1. Overview




*Class diagram 76: OpenHealthCardApp*

## 8.8.2. Getting Started

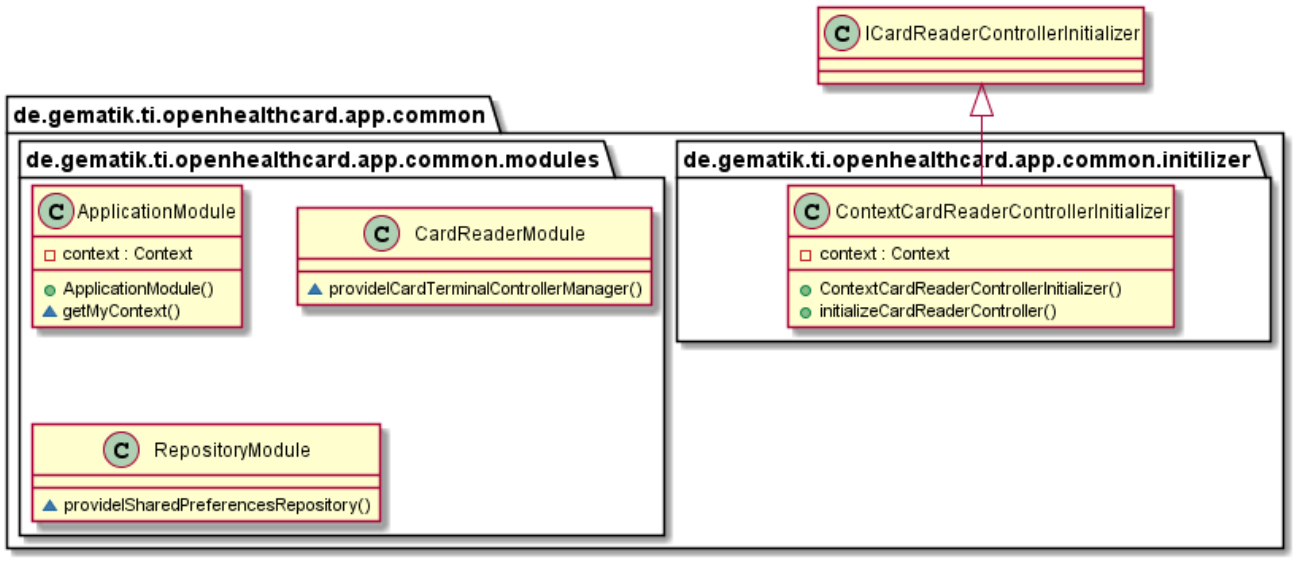
Google PlayStore "OpenHealthCard-Demo" https://play.google.com/store/apps/details?id=de.gematik.ti.openhealthcard.app

## 8.8.3. Open Health Card Android Application Common Library

This part describes the structure and functionality of Common Library in OpenHealthCard App for Android

### 8.8.3.1. Structure

#### 8.8.3.1.1. Overview




*Class diagram 77: Common Library for OpenHealthCard Application*

#### 8.8.3.1.2. Modules

This Modules are used to initialize the Dagger2 ApplicationComponent in Android Application with the needed data objects.

##### 8.8.3.1.3. Initializer

Implementation if a initializer to setup the CardReaderController and inject the android application context if needed.

#### 8.8.3.2. Getting Started

##### 8.8.3.2.1. Build setup

The common library for OpenHealthCard Application contains application module across functionalities, you need just to include following dependency:

*Listing 43. Gradle dependency settings to use Common library for OpenHealthCard Application*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'openhealthcard.app.common', version:
'0.+'
}
```

*Listing 44. Maven dependency settings to use OpenHealthCard Common library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>openhealthcard.app.common</artifactId>
        <version>0.+</version>
    </dependency>
</dependencies>
```

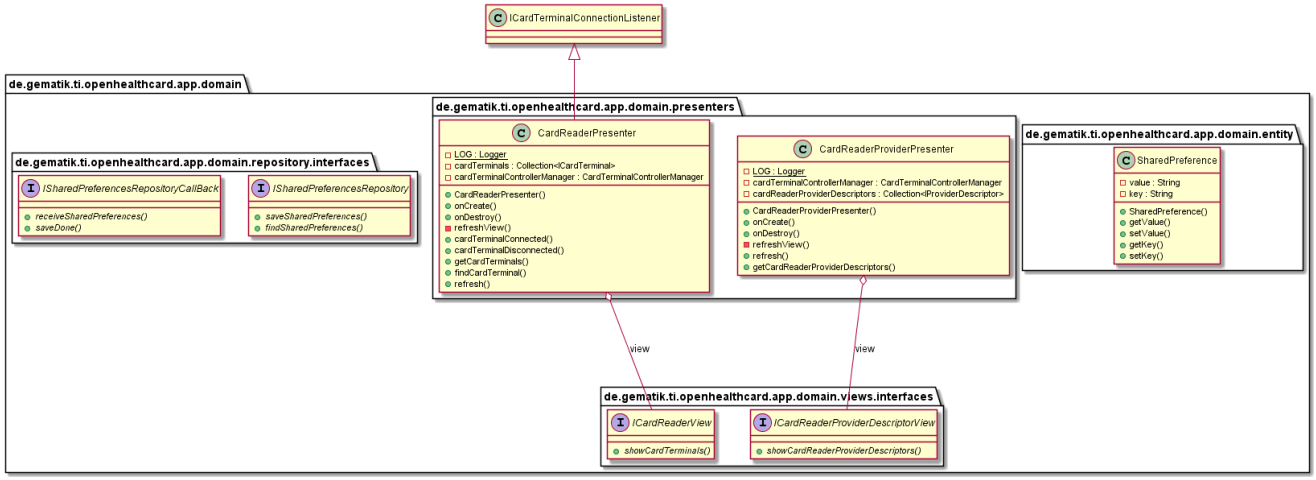
## 8.8.4. Open Health Card Android Application Domain Library

This part describes the structure and functionality of Domain Library OpenHealthCard App for Android

#### 8.8.4.1. Structure

##### 8.8.4.1.1. Overview

The domain layer is devided into interfaces für views and repositories, presenter to process the data before showing and entity objects




*Class diagram 78: Domain Library for OpenHealthCard Application*

#### 8.8.4.1.2. Presenter

The presenter receive the data and show the data on the registered views. This View are card reader views card reader descriptor views.

### 8.8.4.2. Getting Started

#### 8.8.4.2.1. Build setup

The domain library for OpenHealthCard Application contains the domain specific functionalities, you need just to include following dependency:

*Listing 45. Gradle dependency settings to use Domain library for OpenHealthCard Application*

```
dependencies {
    implementation group: 'de.gematik.ti', name: 'openhealthcard.app.domain', version:
'0.+'
}
```

*Listing 46. Maven dependency settings to use OpenHealthCard Domain library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>openhealthcard.app.domain</artifactId>
        <version>0.+</version>
    </dependency>
</dependencies>
```

## 8.8.5. Open Health Card Android Application Card Reader Provider Library

This part describes the structure and functionality of CardReaderProvider Library for OpenHealthCard App for Android

#### 8.8.5.1. Structure

##### 8.8.5.1.1. Overview


*Class diagram 79: Card Reader Library for OpenHealthCard Application*

#### 8.8.5.2. Getting Started

##### 8.8.5.2.1. Build setup

The card reader provider library for OpenHealthCard Application contains the supported card reader provider, you need just to include following dependency:

*Listing 47. Gradle dependency settings to use Card Reader Provider library for OpenHealthCard Application*

```
dependencies {
    implementation group: 'de.gematik.ti', name:
'openhealthcard.app.cardreader.provider', version: '0.+'
}
```

*Listing 48. Maven dependency settings to use OpenHealthCard Card Reader Provider library*

```
<dependencies>
    <dependency>
        <groupId>de.gematik.ti</groupId>
        <artifactId>openhealthcard.app.cardreader.provider</artifactId>
        <version>0.+</version>
    </dependency>
</dependencies>
```