Datatrans Payment Library for Android

Developer's Manual

Datatrans AG
Swiss E-Payment Competence
Kreuzbühlstrasse 26, 8008 Zürich, Switzerland
Tel. +41 44 256 81 91, Fax +41 44 256 81 98
www.datatrans.ch

Revisions

Version	Date	Author	Comment
0.1	2011-02-08	Basil Achermann ieffects ag	First draft
1.0	2011-02-15	Basil Achermann	1.0
1.1	2011-12-15	Basil Achermann	1.1 (PostFinance support)
1.2	2013-01-15	Basil Achermann	1.2 (Maintenance release)
1.3	2013-10-29	Basil Achermann	MyOne added
1.4	2014-02-13	Basil Achermann	Payment options, Android 4.4 fixes
1.5	2014-04-17	Basil Achermann	Alias generation in standard mode; PayPal, PostFinance Card recurring payments
1.5.1	2014-07-03	Basil Achermann	Certificate pinning option
1.5.2	2014-08-13	Basil Achermann	Swisscom Easypay added
1.6.0	2014-09-12	Basil Achermann	Alias generation in hidden mode
1.6.1	2014-11-27	Basil Achermann	PostFinance Card registration
1.6.2	2015-01-09	Basil Achermann	Target SDK 21 support (Android 5)
1.7.0	2015-03-16	Basil Achermann	Easypay Alias support, Lastschrift (ELV) method added, context abstraction (DisplayContext)
1.7.1	2015-04-02	Basil Achermann	ELV aliases with bankrouting
1.7.2	2015-04-16	Basil Achermann	Language fix
1.8.0	2015-07-17	Basil Achermann	SwissBilling added
1.9.0	2015-10-16	Basil Achermann	JCB added
2.0.0	2015-10-29	Basil Achermann	TWINT added
2.0.1	2016-06-29	Basil Achermann	TWINT with new payment pages
2.1.0	2016-07-14	Patrick Schmid	TWINT alias support
2.1.1	2016-08-19	Patrick Schmid	TWINT alias adjustments
2.1.2	2016-09-27	Patrick Schmid	Recurring payment methods returned if authorization skipped
2.2.0	2016-11-16	Patrick Schmid	Currency on alias request, switch to backup URL, TWINT fix
2.2.1	2017-01-09	Patrick Schmid	Discover added, payment method on BusinessException
2.2.2	2017-03-24	Patrick Schmid	Payment with amount > 0 only
2.3.0	2017-07-07	Patrick Schmid	Reka added, merchant properties always sent to post URL
2.4.0	2017-08-25	Patrick Schmid	TWINT payments supported again
2.5.0	2017-09-29	Nathanaël Mägli	TWINT alias support added (TWINT User on File)
2.6.0	2018-01-31	Nathanaël Mägli	Partial TLS 1.2 support for API 16-19
2.7.0	2018-03-15	Patrick Schmid	Enhanced PayPal security
2.8.0	2018-04-25	Patrick Schmid	PostFinance Card expiry, Diners CVV Fix
2.8.1	2018-05-03	Nathanaël Mägli	Added UATP payment method

2.8.2	2018-05-18	Nathanaël Mägli	Improved parameter decoding
2.8.3	2018-07-10	Nathanaël Mägli	Bugfix credit card autofill
2.8.4	2018-07-18	Nathanaël Mägli	Reka Rail & Reka Lunch support
2.8.5	2018-08-14	Patrick Schmid	Replaced PaymentMethodSwissBilling by option
2.9.0	2018-08-30	Patrick Schmid	Samsung Pay and Byjuno direct invoice added
3.0.0	2018-09-14	Patrick Schmid	Added Google Pay
3.1.0	2018-12-06	Patrick Schmid	Deferred authorization
3.1.1	2019-01-28	Patrick Schmid	Samsung Pay API changes
3.1.2	2019-02-15	Nathanaël Mägli	Settlement step added to payment process
3.1.3	2019-02-22	Patrick Schmid	TWINT reqtype fix
3.2.0	2019-04-09	Patrick Schmid	Detailed error codes, switched to AAR packaging, uniform merchant properties, Samsung Pay service IDs
3.3.0	2019-07-02	Patrick Fompeyrine	Added SwissPass, refactored Address & Customer objects, recurring payment methods to/from JSON string, Bugfixes
3.4.0	2019-07-25	Melanie Hüsser	Added SwissPass alias and POWERPAY
3.5.0	2019-08-13	Patrick Fompeyrine	3D secure when requesting alias, verify credit card alias
3.5.1	2019-11-04	Patrick Fompeyrine	Removed customer option requirement for alias payment
3.5.2	2019-12-12	Patrick Fompeyrine	Improved activity lifecycle handling
3.5.3	2020-02-25	Patrick Fompeyrine	Workaround of Android 5 WebView bug
4.0.0	2020-07-01	Patrick Fompeyrine	Support for new backend JSON API flow
4.0.1	2020-07-09	Patrick Fompeyrine	Bugfixes
4.1.0	2020-07-28	Patrick Fompeyrine	Added Coop Supercard
4.1.1	2020-08-21	Patrick Fompeyrine	API 30 release, support createConfigurationContext
4.2.0	2020-09-21	Patrick Fompeyrine	Added Paysafecard, Bugfixes
4.2.1	2020-10-09	Patrick Fompeyrine	Added merchant properties on all TWINT calls
4.2.2	2020-10-27	Patrick Fompeyrine	Window leak crash fix
4.3.0	2020-11-03	Patrick Fompeyrine	Support PostFinance app switch
4.3.1	2020-11-25	Patrick Fompeyrine	Adapting to backend API change
4.3.2	2020-11-27	Patrick Fompeyrine	Use refno from backend JSON API flow for alias requests
4.3.3	2020-12-08	Nathanaël Mägli	TWINT alias request returns transaction Id
4.3.4	2021-01-13	Patrick Fompeyrine	Improve stability of external web process

Table of Contents

1 Introduction 6

1.1 Document Structure 6
1.2 Scope 6
1.3 Conventions 6

2 Overview 8

2.1 Payment Methods 8
2.2 Supported Platforms 8
2.3 Library Tasks 8
2.4 Payment Process 8
2.5 User Interface 9

3 Key Concepts 10

3.1 PaymentProcessAndroid 10
3.2 Library Invocation 10
3.3 State Notification 12
3.4 Recurring payments 13
3.4.1 (De-)Serialization to/from JSON of recurring payment method 13
3.5 Payment method registration (alias request) 13
3.5.1 Payment method selection/input by library (standard mode) 13
3.5.2 Payment method preselected by app, input by library 14
3.5.3 Credit card selection/input by app (hidden mode) 14
3.6 Deferred Authorization 15
3.7 Merchant Notification 16
3.8 Error Handling 16
3.8.1 Technical Errors 17
3.8.2 Business Errors 17
3.8.3 SSL Errors 17
3.9 New JSON API Flow 17
3.9.1 Credit card selection/input by app (hidden mode) 18

4 Mandatory settings 19

4.1 TWINT 19
4.1.1 TWINT not installed error 19
4.2 PayPal 19
4.3 SwissBilling 20
4.4 Byjuno direct invoice 20
4.5 SwissPass 20
4.6 POWERPAY 21
4.7 Paysafecard 21
4.8 Samsung Pay 21
4.8.1 Configure apps 21
4.8.2 Supported Networks (Cards) 21

4.8.3 Regular Payment 22
4.8.4 Samsung Pay Button 22
4.9 Google Pay 22
4.9.1 Configure app for Google Pay 22
4.9.2 Supported Networks (Cards) 22
4.9.3 Regular Payment 23
4.9.4 Google Pay Button 23
4.9.5 Going Live 23
4.10 Recurring payment methods (de-)serialization to/from JSON 23

5 API 24

6 Library Integration 25

6.1 Package Contents 25
6.2 Android Studio Integration 25
6.3 Proguard/R8 rules 25

7 Appendix 27

7.1 List of Illustrations 27
7.2 List of Code Listings 27
7.3 List of Tables 27

1 Introduction

Datatrans AG, leading Swiss payment service provider, developed *Datatrans iOS Payment Library*, allowing application developers to easily integrate Datatrans AG's payment services natively on the iPhone and iPad. Following its success, a version for Android-based devices has been developed.

This manual provides guidance on library installation, invocation, and other issues of importance to developers wishing to integrate *Datatrans Payment Library* (*DTPL*) for Android into their mobile applications.

1.1 Document Structure

Chapter 1 – Introduction
Explains this document's structure and content.

Chapter 2 – Overview
Gives an overview of the Datatrans Payment Library for Android.

Chapter 3 – Key Concepts
Explains key concepts of DTPL for Android and discusses some of the most common use cases.

Chapter 4 – API
Gives an overview over the library's classes.

Chapter 5 – Integration
Explains library installation and integration into Eclipse/ADT.

1.2 Scope

This document provides information on using DTPL to create mobile commerce apps on Android devices. As such, it is primarily aimed at developers.

It is assumed that the reader is already familiar with Datatrans AG's products and services. Also, knowledge of the *Java* programming language, *Android SDK*, as well as basic understanding of *Eclipse* and the *ADT* plugin are required.

Detailed description of the library's API is not part of this document. Javadoc documentation is provided in a separate directory.

1.3 Conventions

Throughout this document, the following styles are used:

Name
Emphasized technical terms, organization/product names

Path
File system paths, file names etc.

Class
Class and method names

```
void codeSample() {
   code(); // sample code
}
```

Code listings

<replaceable>
Text meant to be replaced with data by the developer

2 Overview

2.1 Payment Methods

The library currently supports the following credit cards: VISA, MasterCard, Diners Club, American Express, JCB, UATP, Manor MyOne, Discover and Coop Supercard. Additionally, PayPal, PostFinance Card/E-finance, Swisscom Easypay, Lastschrift (ELV), SwissBilling, Samsung Pay, Byjuno direct invoice, Google Pay, TWINT, Reka, SwissPass, POWERPAY as well as Paysafecard are supported.

2.2 Supported Platforms

Android devices with OS 4.1 (Jelly Bean, API level 16) or higher are supported. The library has been localized for English, French, German, Italian, and Dutch.

2.3 Library Tasks

The payment library is responsible for the following tasks:

- **Validation:** credit card number, expiration date and CVV are validated online.
- **Authentication:** if merchant and credit card are enrolled with 3-D Secure services, authentication ensures that the card is being used by its legitimate owner.
- **Authorization:** if amount and currency are valid and within the card's limit, the payment transaction is authorized and can be completed by the merchant once goods are being delivered (settlement process).

2.4 Payment Process

Figure 2-1 gives an overview of the shopping and payment process on the mobile phone.

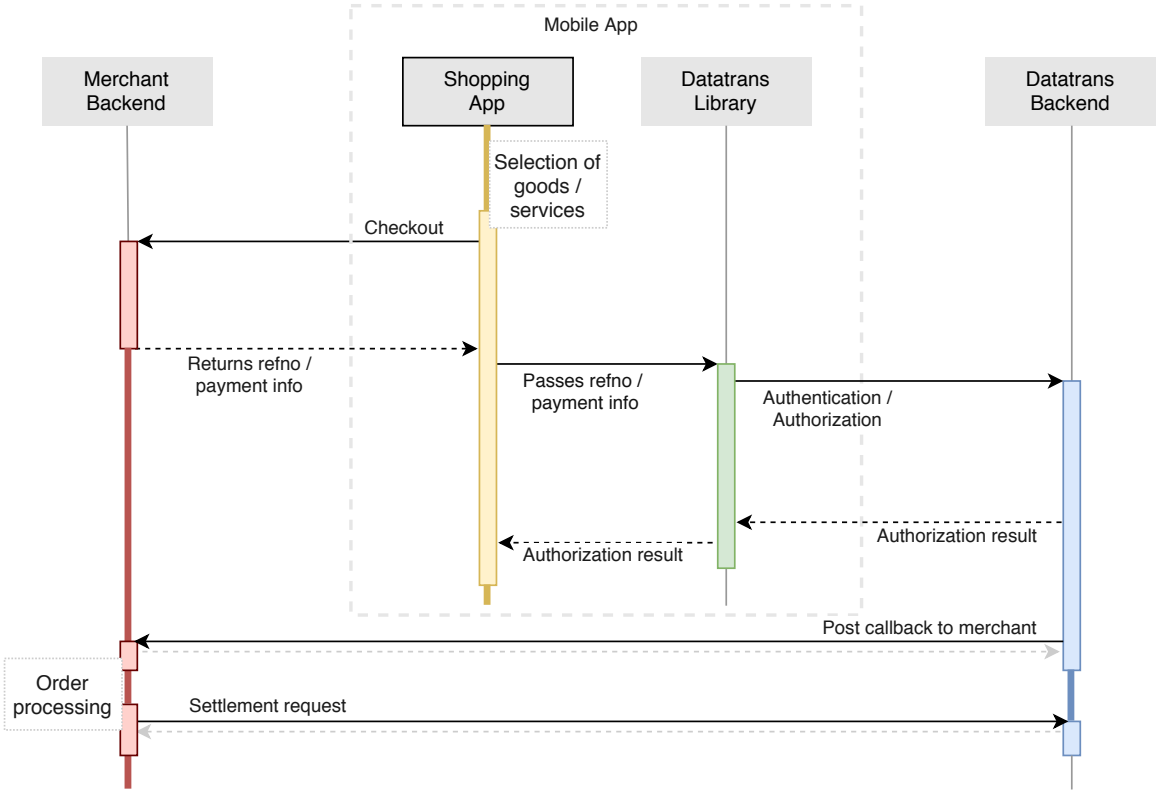

Figure 2-1: Payment process overview

The following steps occur during a successful session:

1. Host app: user selects goods/services to buy from a merchant. When the user proceeds to checkout, complete order information is sent to the merchant's server. In return, the app receives a transaction reference number (*refno)*.
2. App passes payment information and refno to DTiPL.
3. In a series of network calls the library performs all necessary steps to authenticate the user (including 3-D Secure) and authorize the purchase.
4. Transaction is authorized in the background.
5. When authorization is completed, the merchant's server is informed by Datatrans AG's server. The previously supplied *refno* (see step 1) is used to identify and execute the order.
6. App control is given back to the main app component via callback.
7. Merchant server makes settlement request to Datatrans server.

2.5 User Interface

The payment library does not come with a user interface other than a web view. The web view can be presented either in full screen format or embedded into an existing app screen.

3 Key Concepts

3.1 PaymentProcessAndroid

The library's core component is the PaymentProcessAndroid class. The process can be started with or without prior selection of a payment method. If no payment method is supplied, the process starts with a selection web page in full screen format. The user can cancel the process by pressing the back button present on all Android devices.

If payment method is selected by the app, the web view can be displayed either full screen or in an Android ViewGroup. In the latter case, the application carries the responsibility for screen design, controls (e.g. cancel button, hardware back button) or other UI elements. The library just plugs its web view into the view group.

3.2 Library Invocation

Prior to library invocation, the host app must obtain a unique transaction reference number (*refno*) to identify the order. This is typically done by sending complete order information (basket contents, shipping information etc.) to the merchant's web server. The server generates a refno that is stored along with the order and sends it back to the device. Optionally, the server also returns the HMAC-SHA256 signature for additional payment security.

The library is invoked with refno, merchant ID, pricing information and one or several payment methods. If multiple payment methods are supplied, a full-screen, web-based selection screen is displayed (*Standard mode*).

Listing 3-1 shows an example of how DTPL is invoked with several payment methods.

```
String merchantId = "12345"; // Datatrans merchant ID
String refno = "refno12345"; // supplied by merchant's server
String currencyCode = "CHF";
int amount = 1000; // 10.-
String signature = null;

Payment payment = new Payment(merchantId, refno, currencyCode,
                              amount, signature);

ArrayList<PaymentMethod> methods = new ArrayList<>();
// Add here all available payment methods, e.g.
methods.add(new PaymentMethod(PaymentMethodType.VISA));
methods.add(new PaymentMethod(PaymentMethodType.PAYPAL));
methods.add(new PaymentMethod(PaymentMethodType.PFCARD));

DisplayContext dc = new DisplayContext(new ResourceProvider(), context);
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, payment, methods);
ppa.setTestingEnabled(true);
ppa.addStateListener(myListener);
ppa.start();
```

Listing 3-1: Payment process invocation with several payment methods (Standard mode)

Some notes:

- The process in this example is started in test mode. No actual payments can be made. Test mode is off by default.
- context is the Android app context

- No signature is used in this example.

If, on the other hand, the payment method has been previously determined, payment takes place with little or no user interaction. Listing 3-2 shows an example of how DTPL is invoked in a view group with a given payment method.

```
String merchantId = "12345"; // Datatrans merchant ID
String refno = "refno12345"; // supplied by merchant's server
String currencyCode = "CHF";
int amount = 1000; // 10.-

Payment payment = new Payment(merchantId, refno, currencyCode,
                              amount, null);

PaymentMethodCreditCard pm = new PaymentMethodCreditCard(
                PaymentMethodType.VISA, "4900000000000003", 2021, 12,
                123, "Max Muster"); // pay by VISA
// PaymentMethod pm = new PaymentMethod(PaymentMethodType.PAYPAL);// or PayPal
// PaymentMethod pm = new PaymentMethod(PaymentMethodType.PFCARD);// PostFinance
// PaymentMethod pm = new PaymentMethod(PaymentMethodType.VISA);// or VISA
// AliasPaymentMethod pm = new AliasPaymentMethodCreditCard(
//              PaymentMethodType.VISA, "61219152351000133", "",
//              2021, 12, "Max Muster"); // or VISA alias

ViewGroup viewGroup = (ViewGroup)findViewById(R.id.paymentContainer);

DisplayContext dc = new DisplayContext(new ResourceProvider(), viewGroup);
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, payment, pm);

ppa.setTestingEnabled(true);
ppa.addStateListener(myListener);
ppa.start();
```

Listing 3-2: Payment process invocation with preselected payment method

Table 3-1 lists all payment methods that can be used in this mode.

PaymentMethodType	PaymentMethod Class	Description
VISA, MASTERCARD, DINERS, AMEX, JCB, UATP, MYONE, DISCOVER, SUPERCARD	PaymentMethod	Credit card (standard mode)
VISA, MASTERCARD, DINERS, AMEX, JCB, UATP, MYONE, DISCOVER, SUPERCARD	PaymentMethodCreditCard	Credit card (hidden mode)
VISA, MASTERCARD, DINERS, AMEX, JCB, UATP, MYONE, DISCOVER, SUPERCARD	AliasPaymentMethodCreditCard	Credit card (alias/recurring payment)
PFEFINANCE	PaymentMethod	PostFinance E-finance
PFCARD	PaymentMethod	PostFinance Card
PAYPAL	PaymentMethod	PayPal
EASYPAY	PaymentMethod	Swisscom Easypay
ELV	PaymentMethod	Lastschrift

TWINT	PaymentMethod	TWINT
REKA	PaymentMethod	Reka
SWISSBILLING	PaymentMethod	SwissBilling
BYJUNO_DIRECT_INVOICE	PaymentMethod	Byjuno direct invoice
SAMSUNG_PAY	PaymentMethod	Samsung Pay
GOOGLE_PAY	PaymentMethod	Google Pay
SWISSPASS	PaymentMethod	SwissPass
POWERPAY	PaymentMethod	POWERPAY
PAYSAFECARD	PaymentMethod	Paysafecard
EASYPAY, TWINT, SWISSPASS, POWERPAY	AliasPaymentMethod	Alias payment
PFCARD	AliasPaymentMethodPostFinanceCard	PostFinance alias payment
ELV	AliasPaymentMethodELV	Lastschrift alias payment
REKA	AliasPaymentMethodReka	Reka alias payment
PAYPAL	AliasPaymentMethodPayPal	PayPal alias payment

Table 3-1: Supported app-selected payment methods

3.3 State Notification

The app must register a listener implementing the IPaymentProcessStateListener interface. The initial state is NOT_STARTED. The final state is COMPLETED if payment was successful, ERROR or CANCELED if it was not. Listing 3-3 shows a sample listener implementation.

```
@Override
public void paymentProcessStateChanged(PaymentProcessAndroid process) {
   switch (process.getState()) {
      case COMPLETED:
         AliasPaymentMethod pm = process.getAliasPaymentMethod();
         if (pm != null) {
            // serialize and securely store pm for reuse
         }
         break;
      case CANCELED:
         // ignore, abort checkout, whatever...
         break;
      case ERROR:
         Exception e = process.getException();
         if (e instanceof BusinessException) {
            BusinessException be = (BusinessException)e;
            int errorCode = be.getErrorCode(); // Datatrans error code if needed
            // display some error message
         } else {
            // unexpected technical exception, either fatal TechnicalException or
            // javax.net.ssl.SSLException (certificate error)
         }
         break;
   }
}
```

Listing 3-3 Listener notification

Please note that notifications are synchronously performed on the thread responsible for the state change. This is not necessarily the UI-thread. UI-actions should therefore be posted to the UI-thread using android.os.Handler.

3.4 Recurring payments

The library supports recurring payments for credit card, PayPal, PostFinance Card, Easypay, Lastschrift, Reka, TWINT, SwissPass and POWERPAY payments. If recurring payments are enabled, the app can retrieve a AliasPaymentMethod at the end of a successful transaction (see Listing 3-3) or at the end of a card registration process (see section 3.5) and use this method for subsequent transactions in full hidden mode (except for possible 3-D secure screens). Please note that in order to generate and return an alias at the end of a payment transaction, option setRecurringPayment has to be set to true.

3.4.1 (De-)Serialization to/from JSON of recurring payment method

Alias data returned by the library after a successful transaction or registration needs to be stored for future payments. To facilitate this process and to have a platform independent solution, the library can serialize the AliasPaymentMethod object to a JSON string which, for example, can then be sent to a server or stored locally. If stored locally on the device, appropriate encryption techniques should be applied to protect the data from unauthorized access.

As soon as the user wants to pay with the alias, the previously saved JSON can be deserialized to an AliasPaymentMethod. See Listing 3-4 for an example implementation.

```
String json = aliasPaymentMethod.toJson();
// save the JSON string, e.g. on a server
// ...
// User decides to pay with the alias, retrieve JSON
aliasPaymentMethod = AliasPaymentMethod.fromJson(json);
```

Listing 3-4: (De-)Serialization to/from JSON of an AliasPaymentMethod

Important: Even if an app has its own credit card input dialog it *must never* store the original credit card number or CVV.

3.5 Payment method registration (alias request)

The library supports creating credit card, PostFinance Card, Easypay, Reka, Lastschrift, TWINT, SwissPass and POWERPAY alias identifiers without making a payment. Aliases are allowed to be stored by the app and can be used for future hidden mode payments.

When creating an alias, the app can either use its own card input screen and pass the data to the library or let the library manage payment method input.

3.5.1 Payment method selection/input by library (standard mode)

In this mode, the library presents a web view for payment method selection and input. Credit card data is automatically verified in this mode with a test authorization of a small amount.

Listing 3-5 shows creation of an alias in standard mode on the test system. The app is notified as usual via state listener (alias in process.getAliasPaymentMethod()).

```
DisplayContext dc = new DisplayContext(new ResourceProvider(), appContext);

ArrayList<PaymentMethod> methods = new ArrayList<>();
// Add here all available payment methods, e.g.
methods.add(new PaymentMethod(PaymentMethodType.VISA));
methods.add(new PaymentMethod(PaymentMethodType.PAYPAL));
methods.add(new PaymentMethod(PaymentMethodType.PFCARD));

AliasRequest ar = new AliasRequest(merchantId, currencyCode, methods);
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, ar);

ppa.setTestingEnabled(true);
ppa.addStateListener(PaymentTest.this);

ppa.start();
```

Listing 3-5: Creation of credit card alias in standard mode

3.5.2 Payment method preselected by app, input by library

In this mode, the app invokes the library with a given payment method. The library presents a web view for payment method input. In case of a credit card, the data is automatically verified with a test authorization.

Listing 3-6 shows how a Swisscom Easypay alias is created. The app is notified as usual via state listener (alias in process.getAliasPaymentMethod()).

```
DisplayContext dc = new DisplayContext(new ResourceProvider(), appContext);

AliasRequest ar = AliasRequest(merchantId, currencyCode,
                        new PaymentMethod(PaymentMethodType.EASYPAY));
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, ar);

ppa.setTestingEnabled(true);
ppa.addStateListener(PaymentTest.this);

ppa.start();
```

Listing 3-6: Alias creation with a given payment method

3.5.3 Credit card selection/input by app (hidden mode)

In this mode, the library is invoked with credit card details. The library generates an alias and verifies the given credit card with a test authorization transaction.

Listing 3-7 shows creation of a credit card alias in testing mode. The app is notified as usual via state listener. Note that this example will fail because the given credit card data is not valid.

```
DisplayContext dc = new DisplayContext(new ResourceProvider(), appContext);

PaymentMethodCreditCard pm = new PaymentMethodCreditCard(PaymentMethodType.VISA,
             "4444333322221111", 2021, 12, 123, "Max Muster");
AliasRequest ar = new AliasRequest(merchantId, currencyCode, pm);

PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, ar);
ppa.setTestingEnabled(true);
ppa.addStateListener(PaymentTest.this);

ppa.start();
```

Listing 3-7: Creation of credit card alias in hidden mode

3.6 Deferred Authorization

Sometimes the payment amount is unknown when the user initiates a long-running business transaction. An example would be a check-in / check-out train journey. This can usually be done by server-to-server authorization once the amount is known using a payment method alias.

In case of Google Pay, it is **not** possible to receive an alias for future server-to-server transactions. However, it is possible to obtain a partially authorized transactionId for deferred completion. This process is shown in Figure 3-1, again using a check-in and check-out example. To achieve this, the option skipAuthorizationCompletion needs to be set to true during a regular Google Pay payment. As soon as the amount is known, you can either authorize the payment server-to-server or you invoke the library again. In Listing 3-8, an example implementation is provided.

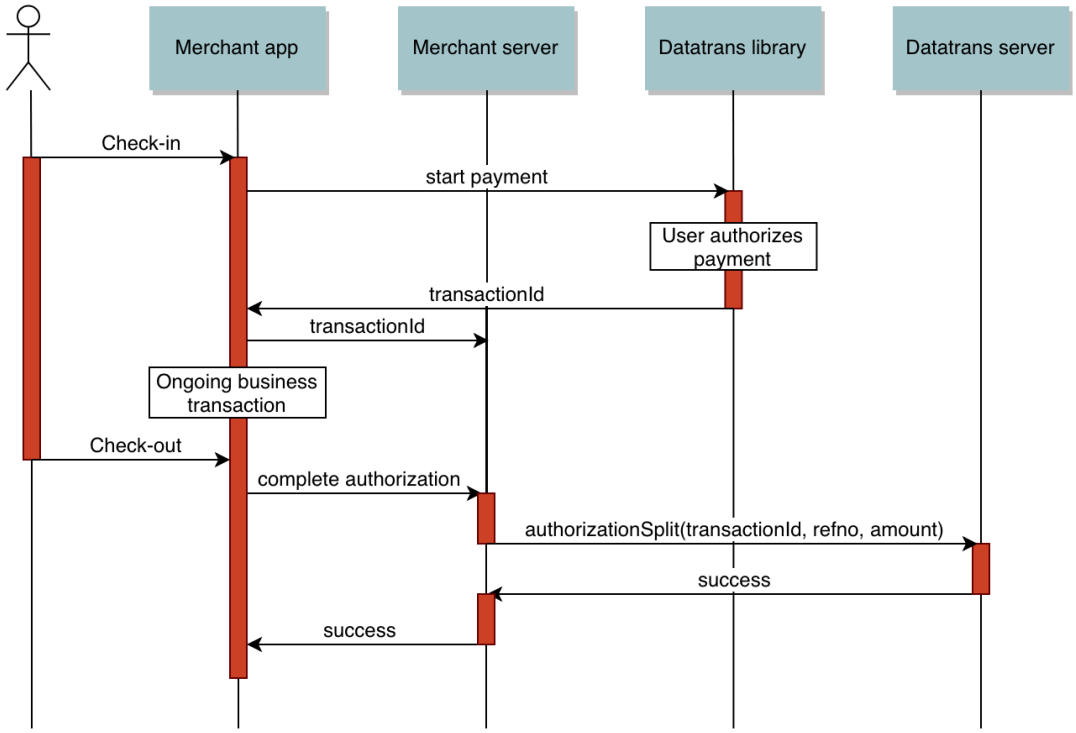

Figure 3-1: Flow of a payment with deferred authorization (Google Pay)

```
// Select / register a payment method
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, aliasRequest);

// -----

// Check-in: in case of Google Pay get a transactionId for future completion
PaymentProcessAndroid ppa;
ppa = new PaymentProcessAndroid(dc, payment, googlePayPaymentMethod);
ppa.getPaymentOptions().setSkipAuthorizationCompletion(true); // etc.
//...
public void paymentProcessStateChanged(final PaymentProcessAndroid process) {
   // get the transaction id and store for actual payment
}

// -----

// Check-out:
// use transaction id, refno and final amount for the actual authorization
// (server-to-server, or
          PaymentProcessAndroid(DisplayContext, PaymentAuthorizationRequest))
```

Listing 3-8: Example implementation of deferred payment authorization (Google Pay)

3.7 Merchant Notification

On successful authorization, Datatrans AG's authorization server invokes the merchant's *postURL* as defined by field URL Post in *Datatrans Web Admin*. Among other information, fields shown in Listing 3-9 are posted as form post or XML post. The merchant's web server retrieves payment information previously stored with the same refno and matches currency code and amount. It then executes the order and performs transaction settlement with Datatrans using the returned uppTransactionId value.

For additional information, please refer to the online documentation at https://docs.datatrans.ch/docs/api-webhooks.

```
amount=1000
currency=CHF
pmethod=VIS
refno=refno12345
uppTransactionId=100916141012915292
acqAuthorizationCode=982889
authorizationCode=915285337
responseCode=01

// if available
aliasCC=70323122544331174
expy=21
expm=12
```

Listing 3-9: postURL fields

3.8 Error Handling

Three kinds of exceptions exist in the library which are treated differently:

- **Technical exceptions:** network interruption, memory or I/O errors
- **Business exceptions:** verification failure, authentication failure, authorization failure. The business exception object may be a generic object of type BusinessException or

a specialized subclass (i.e. TWINTNotInstalledException, see 4.1.1) in order to provide additional information for tailored error messages.

- **SSL exceptions:** An SSLException can occur if the SSL handshake fails in the WebView for API levels 16-18 (TLS 1.2) or if the certificate chain is invalid and pinning is enabled.

3.8.1 Technical Errors

The library is built with the policy that recoverable technical errors lead to non-fatal error messages. The user is encouraged to try again.

3.8.2 Business Errors

The policy for business errors is that the payment process is aborted immediately and no error message is displayed. The exception object can be retrieved from the payment process if it is in state ERROR.

3.8.3 SSL Errors

The WebView on most devices with Android API 16-18 does not support secure connections. In such cases a SSLException is returned.

3.9 New JSON API Flow

In the new JSON API flow, a payment or alias registration is initialized using the new Datatrans backend API (https://api-reference.datatrans.ch/json/#tag/v1transactions). In order to invoke the library, a mobileToken has to be requested in the initialize transaction API call. This is done by adding **returnMobileToken=true** on the OptionRequest. This token can then be used to invoke the PaymentProcessAndroid without providing any payment details.

Note: A new mobile token has to be requested for every invocation of the library.

```
String mobileToken = initializePaymentInBackend();

DisplayContext dc = new DisplayContext(new ResourceProvider(), context);
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, mobileToken);

ppa.setTestingEnabled(true);
ppa.addStateListener(myListener);
ppa.start();
```

Listing 3-10: Invoking the library using the new API flow

Moreover, various parameters from the PaymentOptions object can be added to the initial request to the Datatrans backend. Refer to the online documentation to see which ones are supported.

3.9.1 Credit card selection/input by app (hidden mode)

If the credit card selection and input is handled by the merchant app, the details **must not** be sent in the initial request to the Datatrans backend. Instead, the credit card information should be passed to the PaymentProcessAndroid.

```
String mobileToken = initializePaymentInBackend();

DisplayContext dc = new DisplayContext(new ResourceProvider(), context);
PaymentMethodCreditCard pm = new PaymentMethodCreditCard(PaymentMethodType.VISA,
            "4444333322221111", 2021, 12, 123, "Max Muster");
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, mobileToken, pm);

ppa.setTestingEnabled(true);
ppa.addStateListener(myListener);
ppa.start();
```

Listing 3-11: Hidden mode credit card payment using the new API flow

4 Mandatory settings

4.1 TWINT

4.1.1 TWINT not installed error

If no TWINT app or no up-to-date TWINT app is installed to handle payment or registration instructions, the business exception will be of the type TWINTNotInstalledException.

Please display the following error messages:

Language	Error message
DE	Auf diesem Gerät ist keine oder eine veraltete Version von TWINT installiert. Bitte aktualisieren oder installieren Sie die TWINT App.
EN	No or an outdated version of TWINT is installed on this device. Please update or install the TWINT app.
FR	TWINT n'est pas installée ou une version obsolète de TWINT est installée sur cet appareil. Veuillez mettre à jour ou installer l'app TWINT.
IT	Su questo dispositivo non è installato TWINT, oppure è installata una versione obsoleta. La preghiamo di aggiornare o installare l'App TWINT.

4.2 PayPal

For PayPal payments an external web process is used. After this web process has finished, a callback to your app is issued. In order to receive this callback, you need to define the Datatrans relay activity with an intent filter in your app manifest for a defined scheme as shown in Listing 4-1, **and** configure the setAppCallbackScheme() option (Listing 4-2).

Keep in mind that the URI scheme must be unique to the shopping app and the activity. Do not use actual protocols or file types such as "http", "mailto", "pdf" etc., generic names like "ticket". An example would be the package name extended by an identifier dtpl.

```
<activity
     android:name="ch.datatrans.payment.ExternalProcessRelayActivity"
     android:launchMode="singleTask"
     android:enabled="false"
     android:theme="@android:style/Theme.Translucent.NoTitleBar">
     <intent-filter>
             <action android:name="android.intent.action.VIEW" />

             <category android:name="android.intent.category.DEFAULT" />
             <category android:name="android.intent.category.BROWSABLE" />

             <data android:scheme="your.package.name.dtpl" />
     </intent-filter>
  </activity>
```

Listing 4-1 External process relay activity in manifest

```
paymentProcessAndroid.getPaymentOptions()
                                .setAppCallbackScheme("your.package.name.dtpl");
```

Listing 4-2 Setting app callback scheme on options

Moreover, add the following dependency to the build.gradle of your project:

```
implementation 'androidx.browser:browser:1.3.0'
```

Listing 4-3 Configuration for PayPal in the build.gradle

4.3 SwissBilling

For SwissBilling transactions, a Customer object has to be created and configured via the setCustomer() option. In addition, an optional SwissBillingPaymentInfo can be initialized and set via the setSwissBillingPaymentInfo() option (Listing 4-4). An example of both configurations can be seen in the Javadoc of the SwissBillingPaymentInfo class.

```
paymentProcessAndroid.getPaymentOptions().setCustomer(customer);
paymentProcessAndroid.getPaymentOptions()
                        .setSwissBillingPaymentInfo(swissBillingInfo); // optional
```

Listing 4-4: Configuration for SwissBilling

4.4 Byjuno direct invoice

For Byjuno direct invoice transactions, a Customer object has to be created and configured via the setCustomer() option. In addition, an optional ByjunoPaymentInfo can be initialized and set via the setByjunoPaymentInfo() option (Listing 4-5). An example of both configurations can be seen in the Javadoc of the ByjunoPaymentInfo class.

```
paymentProcessAndroid.getPaymentOptions().setCustomer(customer);
paymentProcessAndroid.getPaymentOptions()
                        .setByjunoPaymentInfo(byjunoInfo); // optional
```

Listing 4-5: Configuration for Byjuno direct invoice

4.5 SwissPass

For SwissPass transactions, there are four possible ways to configure the library:

- Doing nothing
- Provide a Customer object
- Provide a SwissPassPaymentInfo object
- Provide both objects

The Customer and SwissPassPaymentInfo can be set via the setCustomer() or setSwissPassPaymentInfo() option respectively (Listing 4-6). Keep in mind that the user needs to put in the information that are not already provided.

If a Customer is given, then the contents of the object **must** match the details stored in the given SwissPass account. Example configurations can be seen in the Javadoc of the SwissPassPaymentInfo class.

```
paymentProcessAndroid.getPaymentOptions().setCustomer(customer);
paymentProcessAndroid.getPaymentOptions().setSwissPassPaymentInfo(swissPassInfo);
```

Listing 4-6: Configuration for SwissPass

4.6 POWERPAY

For POWERPAY transactions, a Customer object has to be created and configured via the setCustomer() option. In addition to the default parameters, this object must contain a gender and an address with a city. The date of birth can be omitted but the user will be prompted to enter it manually in that case.

```
Address address = new Address("Max", "Muster", "via streccione 2", "6900");
address.setCity("Lugano"); // required
Customer customer = new Customer();
customer.setAddress(address);
customer.setGender("male"); // required
customer.setBirthDate(new Date(1980, 1, 1)); // optional
paymentProcessAndroid.getPaymentOptions().setCustomer(customer);
```

Listing 4-7: Configuration for POWERPAY

4.7 Paysafecard

For Paysafecard transactions, a unique ID has to be set via the setPaysafecardMerchantClientId() option for identifying a customer. As an example, this could be the unique ID of your customer as registered within your database. If you are using the e-mail address or any other personal information, please encrypt it.

```
paymentProcessAndroid.getPaymentOptions().setPaysafecardMerchantClientId(id);
```

Listing 4-8: Configuration for Paysafecard

4.8 Samsung Pay

4.8.1 Configure apps

In order to use Samsung Pay in your test app, you need to add the following to your Android Manifest inside the application tag:

```
<application>
   ...
   <meta-data android:name="debug_mode" android:value="Y" />
   <meta-data android:name="spay_debug_api_key" android:value="" />
   <meta-data android:name="spay_sdk_api_level" android:value="2.3" />
</application>
```

Listing 4-9 Configuration for Samsung Pay in the manifest of the test app

Note: The tag with name spay_debug_api_key must be defined even if it is empty.

For your release app, you only have to add:

```
<application>
   ...
   <meta-data android:name="debug_mode" android:value="N" />
   <meta-data android:name="spay_sdk_api_level" android:value="2.3" />
</application>
```

Listing 4-10 Configuration for Samsung Pay in the manifest of the production app

4.8.2 Supported Networks (Cards)

Samsung Pay must be configured with the list of card types supported by the merchant's acquirer, usually at least Visa and Mastercard. If you would like to support other cards, e.g. American Express, please check with Datatrans support (support@datatrans.ch) or ask your acquirer.

Card types are configured via supportedNetworks parameter as a list of type PaymentMethodType (see code example below).

4.8.3 Regular Payment

If you want to use Samsung Pay just like any other payment method, you have to provide the supported card networks and a name, which will be shown on the Samsung Pay sheet, as shown in Listing 4-11. Furthermore, add SAMSUNG_PAY to the list of payment methods you want to support.

Note that the library determines whether Samsung Pay is present and hides that payment method if the device does not support Samsung Pay.

```
ArrayList<PaymentMethodType> supportedNetworks = new ArrayList<>();
supportedNetworks.add(PaymentMethodType.VISA);
supportedNetworks.add(PaymentMethodType.MASTERCARD);
SamsungPayConfig samsungPayConfig =
   new SamsungPayConfig(supportedNetworks, "Example merchant");
paymentProcessAndroid.getPaymentOptions().setSamsungPayConfig(samsungPayConfig);
```

Listing 4-11 Configure Samsung Pay for payments

4.8.4 Samsung Pay Button

If you want to use a stand-alone Samsung Pay button in your app, please do so by following Samsung Pay's guidelines. However, check first the availability of Samsung Pay by using the SamsungPayAvailabilityChecker (see Javadoc for details). Once the user has pressed the button, configure the payment library as described above and set Samsung Pay as the sole accepted payment method (Listing 4-12). Samsung Pay will then start directly without additional library screens.

```
PaymentMethod pm = new PaymentMethod(PaymentMethodType.SAMSUNG_PAY);
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, payment, pm);
// additional Samsung Pay configurations as explained above...
```

Listing 4-12 Direct invocation of Samsung Pay

4.9 Google Pay

4.9.1 Configure app for Google Pay

In order to use Google Pay in your app, you need to add the following dependency to the build.gradle of your project. Note that you have to explicitly state the two android support libraries as well if there is already another com.android.support package with a specific version in your project, e.g. customtabs.

```
implementation 'com.google.android.gms:play-services-wallet:18.0.0'

// Only needed if there is already a com.android.support library
implementation 'com.android.support:appcompat-v7:<version>'
implementation 'com.android.support:support-v4:<version>'
```

Listing 4-13 Configuration for Google Pay in the build.gradle

4.9.2 Supported Networks (Cards)

Google Pay must be configured with the list of card types supported by the merchant's acquirer, usually at least Visa and Mastercard. If you would like to support other cards, e.g. American Express, please check with Datatrans support (support@datatrans.ch) or ask your acquirer.

Card types are configured via supportedNetworks parameter as a list of type PaymentMethodType (see code example below).

4.9.3 Regular Payment

If you want to use Google Pay just like any other payment method, you have to provide the supported card networks and a name, which will be shown on the Google Pay sheet, as shown in Listing 4-14. Furthermore, add GOOGLE_PAY to the list of payment methods you want to support.

```
ArrayList<PaymentMethodType> supportedNetworks = new ArrayList<>();
supportedNetworks.add(PaymentMethodType.VISA);
supportedNetworks.add(PaymentMethodType.MASTERCARD);
GooglePayConfig googlePayConfig =
                new GooglePayConfig(supportedNetworks, "Example merchant");
paymentProcessAndroid.getPaymentOptions().setGooglePayConfig(googlePayConfig);
```

Listing 4-14 Configure Google Pay for payments

4.9.4 Google Pay Button

If you want to use a stand-alone Google Pay button in your app, please do so by following Google Pay's guidelines. However, check first the availability of Google Pay by using the GooglePayAvailabilityChecker (see Javadoc for details). Once the user has pressed the button, configure the payment library as described above and set Google Pay as the sole accepted payment method (Listing 4-15). Google Pay will then start directly without additional library screens.

```
PaymentMethod pm = new PaymentMethod(PaymentMethodType.GOOGLE_PAY);
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, payment, pm);
// additional Google Pay configurations as explained above...
```

Listing 4-15 Direct invocation of Google Pay

4.9.5 Going Live

There are two important steps that you have to do before using Google Pay in your production app.

- First, you need to request production access from Google Pay Support. This can be done on https://developers.google.com/pay/api/android/guides/test-and-deploy/integration-checklist#requesting-production-access where you also find an integration checklist to ensure everything works as expected.
- Second, you have to enable your app in the Google Pay Developer Profile.

After that, you are ready to request launch approval from Google for your production app.

4.10 Recurring payment methods (de-)serialization to/from JSON

In case that you (de-)serialize recurring payment methods to/from JSON, you need to add the Gson library to your project. Add following line to your build.gradle file:

```
implementation 'com.google.code.gson:gson:2.8.5'
```

Listing 4-16 Configuration for (de-)serializing recurring payment methods to/from JSON in the build.gradle

5 API

Figure 5-1 gives an overview of the library's classes. Full API documentation is located in the javadoc directory of the documentation folder.

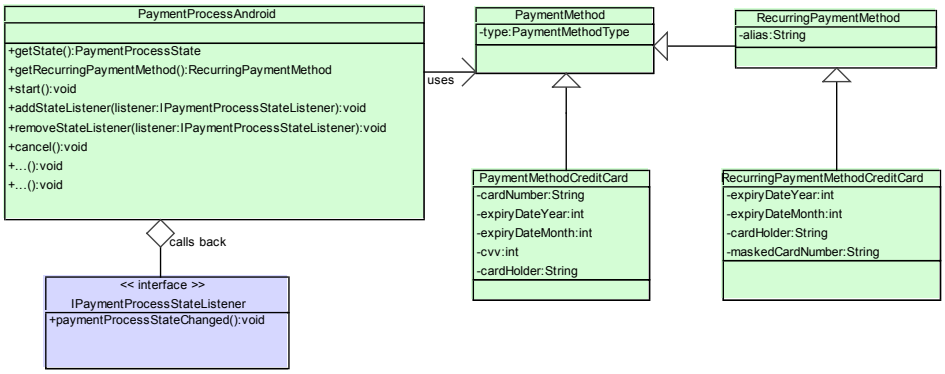

Figure 5-1: Library classes

6 Library Integration

6.1 Package Contents

The library is distributed as a single .zip file with a directory structure shown in Table 6-1.

Directory	Description
/doc	Contains this documentation and Javadoc.
/dtapl-X.X.X.aar	AAR file to be added to Android Studio

Table 6-1: Directory structure

6.2 Android Studio Integration

The easiest way to integrate a local AAR file to a project is to use the module import wizard. This can be found under File->New Module... and then choose Import .JAR/.AAR Package (Figure 6-1). After that, you only have to point to the AAR file and Android Studio creates a library module for it.

You can now use the library in your project.

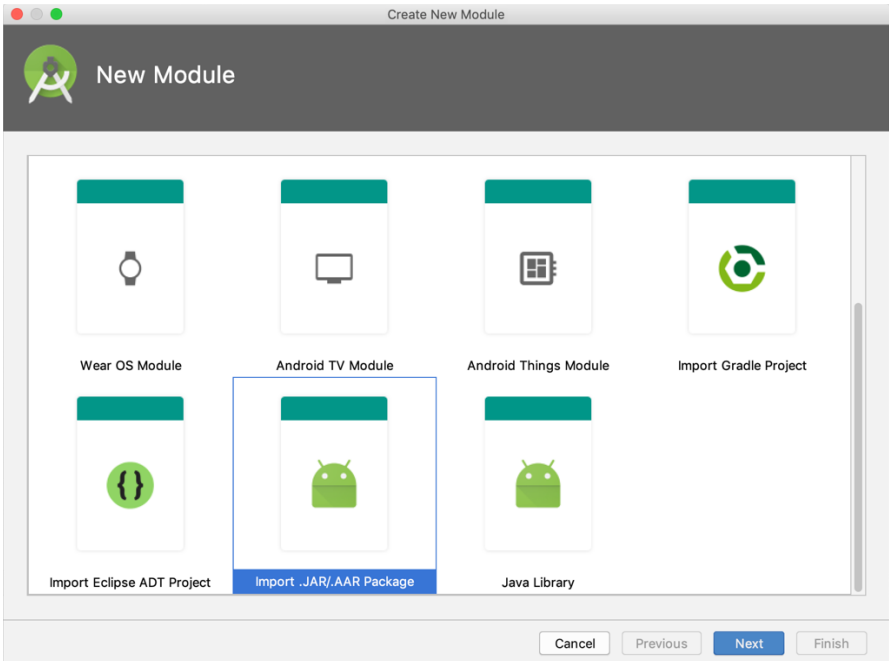

Figure 6-1: Import wizard for AAR package

6.3 Proguard/R8 rules

This part of the integration is needed in two cases:

- When app stores alias payment methods as serialized data
- Samsung Pay

The purpose of these rules for Proguard/R8 is to remain compatible with future versions. Add the needed rules in Listing 6-1 to your proguard-rules.pro file.

```
# needed when using serialized alias payment methods
-keepnames class * implements java.io.Serializable
-keepclassmembers class * implements java.io.Serializable {
    static final long serialVersionUID;
    private static final java.io.ObjectStreamField[] serialPersistentFields;
    !static !transient <fields>;
    private void writeObject(java.io.ObjectOutputStream);
    private void readObject(java.io.ObjectInputStream);
    java.lang.Object writeReplace();
    java.lang.Object readResolve();
}

-keep class com.samsung.** { *; } # needed for Samsung Pay
```

Listing 6-1 Proguard/R8 rules

7 Appendix

7.1 List of Illustrations

Figure 2-1: Payment process overview 8
Figure 3-1: Flow of a payment with deferred authorization (Google Pay) 15
Figure 5-1: Library classes 24
Figure 6-1: Import wizard for AAR package 25

7.2 List of Code Listings

Listing 3-1: Payment process invocation with several payment methods (Standard mode) 10
Listing 3-2: Payment process invocation with preselected payment method 11
Listing 3-3 Listener notification 13
Listing 3-4: (De-)Serialization to/from JSON of an AliasPaymentMethod 13
Listing 3-5: Creation of credit card alias in standard mode 14
Listing 3-6: Alias creation with a given payment method 14
Listing 3-7: Creation of credit card alias in hidden mode 14
Listing 3-8: Example implementation of deferred payment authorization (Google Pay) 16
Listing 3-9: postURL fields 16
Listing 3-10: Invoking the library using the new API flow 17
Listing 3-11: Hidden mode credit card payment using the new API flow 18
Listing 4-1 External process relay activity in manifest 19
Listing 4-2 Setting app callback scheme on options 20
Listing 4-3 Configuration for PayPal in the build.gradle 20
Listing 4-4: Configuration for SwissBilling 20
Listing 4-5: Configuration for Byjuno direct invoice 20
Listing 4-6: Configuration for SwissPass 20
Listing 4-7: Configuration for POWERPAY 21
Listing 4-8: Configuration for Paysafecard 21
Listing 4-9 Configuration for Samsung Pay in the manifest of the test app 21
Listing 4-10 Configuration for Samsung Pay in the manifest of the production app 21
Listing 4-11 Configure Samsung Pay for payments 22
Listing 4-12 Direct invocation of Samsung Pay 22
Listing 4-13 Configuration for Google Pay in the build.gradle 22
Listing 4-14 Configure Google Pay for payments 23
Listing 4-15 Direct invocation of Google Pay 23
Listing 4-16 Configuration for (de-)serializing recurring payment methods to/from JSON in the build.gradle 23
Listing 6-1 Proguard/R8 rules 26

7.3 List of Tables

Table 3-1: Supported app-selected payment methods 12
Table 6-1: Directory structure 25