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

Table of Contents

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, and Manor MyOne. Additionally, PayPal, PostFinance Card/E-finance, Swisscom Easypay, Lastschrift (ELV), SwissBilling, as well as TWINT are supported.

2.2 Supported Platforms

Android devices with OS 2.2 (API level 8) 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, 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.

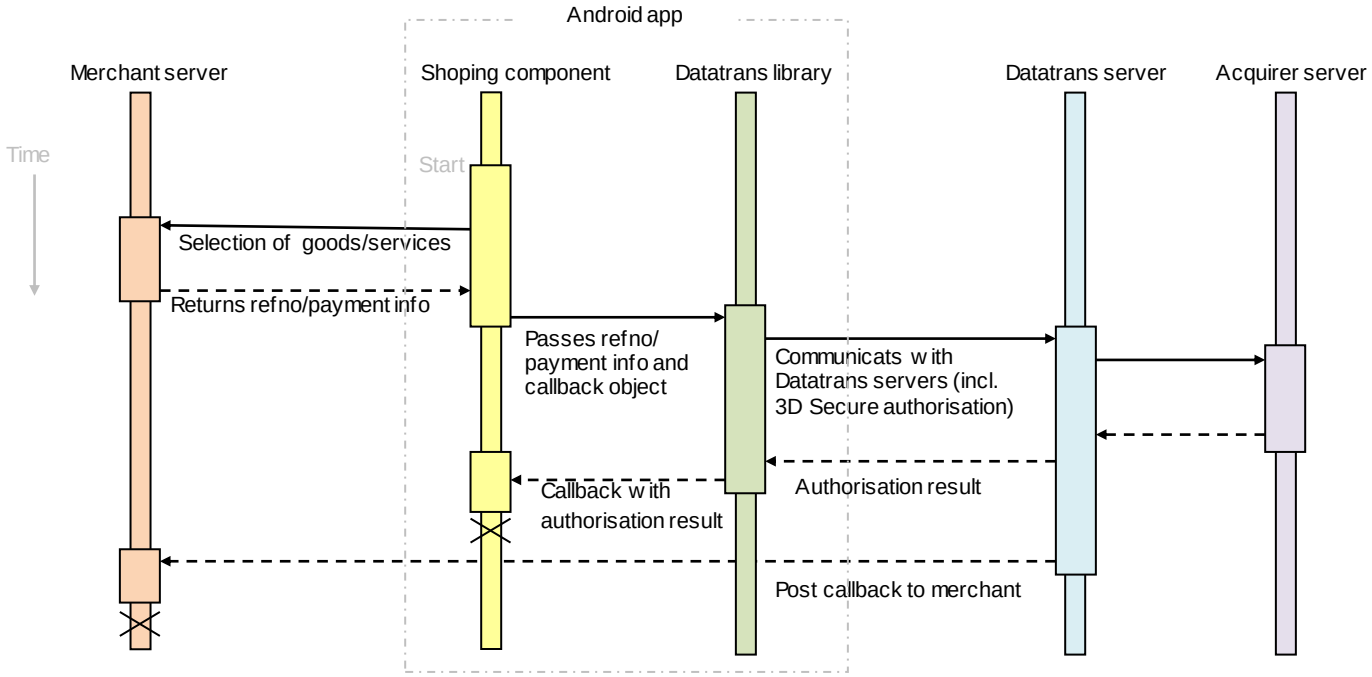

Figure 2-1: Payment process overview

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.

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-MD5 signature for additional payment security.

The library is invoked with refno, merchant ID, pricing information and an optional payment method. If no payment method is supplied, a full-screen, web-based selection screen is displayed (*Standard mode*).

Listing 3-1 shows an example of how DTPL is invoked without payment method.

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

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

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

Listing 3-1: Payment process invocation without payment method (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, null);

PaymentMethodCreditCard pm = new PaymentMethodCreditCard(
                PaymentMethodType.VISA, "4900000000000003", 2015, 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", "",
//              2015, 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, MYONE	PaymentMethod	Credit card (standard mode)
VISA, MASTERCARD, DINERS, AMEX, JCB, MYONE	PaymentMethodCreditCard	Credit card (hidden mode)
VISA, MASTERCARD, DINERS, AMEX, JCB, MYONE	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
PAYPAL, EASYPAY	AliasPaymentMethod	Alias payment
PFCARD	AliasPaymentMethodPostFinanceCard	PostFinance alias payment
ELV	AliasPaymentMethodELV	Lastschrift alias payment
SWISSBILLING	PaymentMethodSwissBilling	Preselected SwissBilling 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 and Lastschrift 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`.

Typically, an app would use standard Java object serialization/deserialization to store and retrieve recurring payment method information (`AliasPaymentMethod` is `Serializable`). Alias payment method data must be protected from unauthorized access. If stored locally on the device, appropriate encryption techniques should be applied.

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, TWINT and Lastschrift 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.

Listing 3-4 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);

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

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

ppa.start();
```

Listing 3-4: 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.

Listing 3-5 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,
                    new PaymentMethod(PaymentMethodType.EASYPAY), null);
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, ar);

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

ppa.start();
```

Listing 3-5: 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 optionally verifies the given credit card with a test authorization transaction.

Listing 3-6 shows creation of a credit card alias in testing mode with a verifying transaction. The app is notified as usual via state listener. Note that this example will fail because the given credit card data is not valid. The same request would succeed with an unverified AliasRequest.

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

PaymentMethodCreditCard pm = new PaymentMethodCreditCard(PaymentMethodType.VISA,
            "4444333322221111", 2015, 12, 123, "Max Muster");
AliasRequest ar = new AliasRequest(merchantId, pm, true); // true: verified
PaymentProcessAndroid ppa = new PaymentProcessAndroid(dc, ar);
ppa.setTestingEnabled(true);
ppa.addStateListener(PaymentTest.this);

ppa.start();
```

Listing 3-6: Creation of credit card alias in hidden mode

3.6 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 3-7 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 `authorizationCode` value.

```
amount=1000
currency=CHF
refno=refno12345
uppTransactionId=100916141012915292
acqAuthorizationCode=982889
authorizationCode=915285337
```

Listing 3-7: postURL fields

3.7 Error Handling

Two 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 (e.g. `TWINTNotInstalledException`) in order to provide additional information for tailored error messages.

3.7.1 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.

3.7.2 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`.

4 API

Figure 4-1 gives an overview of the library's classes. Full API documentation is located in the `javadoc` directory of the documentation folder.

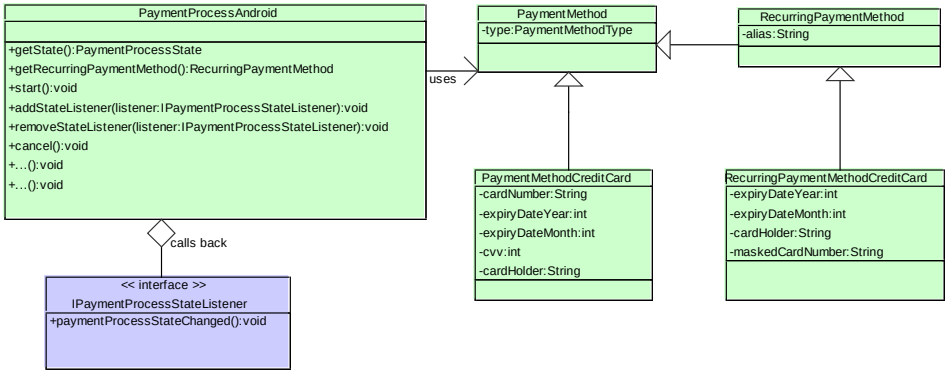

Figure 4-1: Library classes

5 Library Integration

5.1 Package Contents

The library is distributed as a single .zip file with a directory structure shown in Table 5-1.

Directory	Description
`/doc`	Contains this documentation and Javadoc.
`/DatatransPaymentAndroid`	Contains the library project to be added to Eclipse

Table 5-1: Directory structure

5.2 Eclipse Integration

Copy the `DatatransPaymentAndroid` project (whole directory) into your workspace directory.

Open your workspace in Eclipse. Right-click on the `Projects` or `Package Explorer` view and choose `Import…` Select `General->Existing Projects into Workspace`. Click `next`, then `Browse…`, navigate to the `DatatransPaymentAndroid` directory and click `Open`. Click the `Finish` button at the bottom of the dialog.

Create a new Android project if you do not already have a project to work with. Open the project's properties. In the `Android` section, add the `DatatransPaymentAndroid` project as a library. (Figure 5-1)

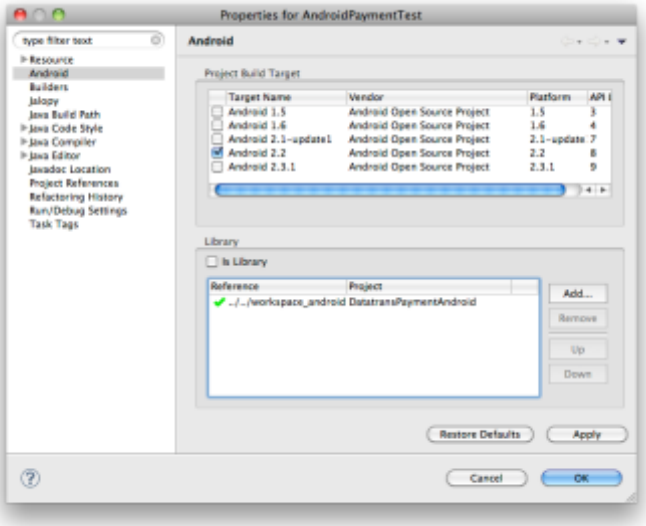

Figure 5-1: Adding the library project

In the `Java Build Path` section, switch to the `Libraries` tab and select `Add JARs…` Navigate to `DatatransPaymentAndroid/libs/` and add the jar file (Figure 5-2). Please check that only the most current version of the library is on the build path. Old versions must be removed.

Make sure you have `android.permission.INTERNET` enabled in your manifest.

You can now use the library in your project.

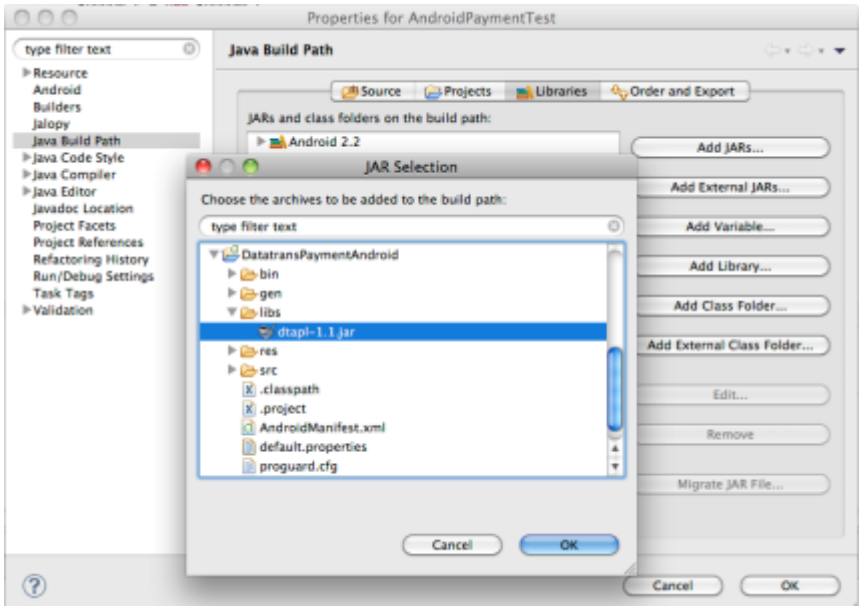

Figure 5-2: Build path configuration

6 Appendix

6.1 References

Number	Document	Version
1		
2		
3		

Table 6-1: List of references to other documents

6.2 List of Illustrations

6.3 List of Code Listings

6.4 List of Tables