Dotpay Technical Support
Wielicka Str. 72, 30-552 Cracow, Poland
phone. +48 12 688 26 00
fax +48 12 688 26 49
e-mail: tech@dotpay.pl

Worldwide secure payment

SDK (Android)

Version 1.2.1

TABLE OF CONTENTS

INTRODUCTION

This document describes a set of software development tools (SDK library) that allows integrating merchant's mobile application with Dotpay payment system.

Thanks to devolving as many steps from our web application as possible to mobile application, the payment process is more convenient for a user and a developer obtains more control over it.

There is an option to modify visualization style of SDK (colors, fonts) to have best integration possible with merchant's mobile application.

The SDK library was created in language Java. It supports Android system version 4.1 (API:16, JELLY BEAN) and higher.

This document uses the following terms and symbols:

Contractor / Merchant	Dotpay service user receiving the payment or owner of web shop, web page, on which payment process starts.
Shop	Merchant's web shop that uses mobile application.
Client / Buyer	The person making the payment to the merchant via the online transaction with a use of mobile application.
Developer	A developer, who creates mobile application for a merchant.

Related documents

Dotpay technical manual – a document that describes a basic payment process for web shops, available for downloading in Dotpay panel.

Getting started with SDK

You have to add the SDK to a project and initialize it in a proper way to use it. Details of these steps were described in next chapters.

In order to make it easier to start with SDK, we deliver a test application, available in `example` sub-directory.

Project settings

1. The SDK library file must be added to the project and is located in `lib` directory.
2. Add Android libraries:

```
compile 'com.android.support:appcompat-v7:25.3.1'
compile 'com.android.support:cardview-v7:25.3.1'
compile 'com.android.support:design:25.3.1'
```

3. Libraries required by SDK have to be added to the project. Add dependencies below to the project:

```
compile 'org.jsoup:jsoup:1.7.2'
compile 'com.squareup.retrofit:retrofit:2.0.2'
compile 'com.squareup.okhttp:okhttp:2.4.0'
compile 'com.squareup.okhttp:okhttp-urlconnection:2.4.0'
compile 'com.mcxiaoke.volley:library:1.0.+'
compile 'com.squareup.picasso:picasso:2.5.2'
compile 'de.greenrobot:eventbus:2.2.0'
compile 'com.j256.ormlite:ormlite-core:4.47'
compile 'com.j256.ormlite:ormlite-android:4.47'
compile 'io.card:android-sdk:5.4.1'
```

   These libraries were added to the test application.

4. You need to initialize SDK before you can use it. Add in `onCreate` method in your `Application` class:

```
AppSDK.initialize(this);
```

Language settings

The SDK library, in a current version, supports Polish and English language. Further language extensions are planned so that dynamic configuration is recommended. In order to do this, you need to:

1. Download available languages list from `PaymentManager`:

```
PaymentManager.getInstance().getLanguages();
```

2. Select most appropriate language
3. Set a selected language

```
PaymentManager.getInstance().setApplicationLanguage(bestLang.getLang());
```

System selection

The SDK library may communicate both with Dotpay production and test environment. With no system selection SDK library won't work properly, therefore it is necessary to select a system.

In order to select the test system, you need to follow the instruction:

```
PaymentManager.getInstance().setApplicationVersion(Configuration.TEST_VERSION);
```

In order to select the production system, you need to follow the instruction:

```
PaymentManager.getInstance().setApplicationVersion(Configuration.RELEASE_VERSION);
```

Additional options

Switching off the last selected channel by a client:

```
Configuration.loadLastSelectedChannel(false);
```

SDK version

We recommend to display the SDK version in the web shop which will make problems diagnostic process easier in the future. The SDK version is available using:

```
Settings.getSDKVersion();
```

Payment process

The payment process consists of handing over a control to the class `PaymentManager` and awaiting an error/success event that returns control.

The SDK library will guide the client through the process of selecting payment channel, passing on/verification client's data, selecting extra options, accepting proper terms of use and finalizing the payment process.

In the event of paying for real goods, payment status received from SDK is just informational, a right order status will be delivered in the backend system according to Dotpay technical manual.

In next chapters we described steps that are required to use `PaymentManager` class and extra options that enable adjusting payment process to own needs.

Registering return callback

Preparing payment process starts from creating callback functions that listen for signalized finish payment events. The `PaymentManagerCallback` interface should be implemented in a class, in which a payment will be initialized.

```
public interface PaymentManagerCallback {
    void onPaymentSuccess(PaymentEndedEventArgs paymentEndedEventArgs);
    void onPaymentFailure(PaymentEndedEventArgs paymentEndedEventArgs);
}
```

And next register it in the PaymentManager:

```
PaymentManager.getInstance().setPaymentManagerCallback(paymentManagerCallback);
```

Payment process initialization

The payment process initialization starts from the use of `initialize` method of `PaymentManager`. This method receives arguments described in a table below. The initialization process has to take place in a current Android activity.

PARAMETER	DESCRIPTION
context	Type: Context Context, from which this method was called out
`paymentInformation`	Typ: `PaymentInformation` Object that contains all required parameters for payment process

An object constructor of type `PaymentInformation` receives arguments in accordance with the following table:

PARAMETER	DESCRIPTION
id	Type: string Merchant's ID number in Dotpay system
amount	Type: double Payment amount
description	Type: string Payment description
currency	Type: string Default value: "PLN" It describes currency of parameter amount. In the chapter *Available currencies* we described a way how to download available currency list.

Additionally, there is possibility to set the following parameters for `PaymentInformation` object, using setters:

PARAMETER	DESCRIPTION
senderInformation	Type: Map<string, string> Default value: null Setter name: `setSenderInformation(Map<String, String> senderInformation)` Additional information on client. Keys: “firstname”; “lastname”; “email”; “phone”; “street"; “street_n1” – building number; "street_n2" – flat number; "postcode"; "city"; "country" (3 letters ISO3166). These values are not mandatory. We recommend to pass on at least name, surname and email. The payment form should be filled out with that type of data. SDK will ask a client for missing data. Specific explanation of these fields is described in Dotpay technical manual.
additionalInformation	Type: Map<string, string> Default value: null

	Setter name: `setAdditionalInformation(Map<String,String> additionalInformation)` Extra parameters handed over in a payment process in accordance with additional technical manuals.
control	Type: string Default value: null Setter name: `setControl(String)` The parameter that defines a payment, handed over in payment confirmation, which is sent to a Shop. This parameter is required to match a payment status to an appropriate order in a Shop. More information you will find in the Dotpay technical manual. If not set, it is generated by SDK. ATTENTION To have properly working payment history, this parameter should be unique for every order.
urlc	Type: string Default value: null Setter name: `setUrlc(String)` URL address used for receiving payment information (order completed or rejected). More information you will find in the Dotpay technical manual.

Example of payment initialization:

```
String description = "zamówienie 12345";
double amount = 123.45;
PaymentInformation paymentInformation = new PaymentInformation(merchant_Id, amount,
description, selectedCurrency);

Map<String, String> sender = new Map<String, String> {{"firstname", "Jan"},
{"lastname", "Kowalski"}, {"email", "jan.kowalski@test.pl"}}
Map<String, String> additional = new Map<String, String> {{"id1", "12345"},
{"amount1", "100"}, {"id2", "67890"}, {"amount2", "23.45"}}

paymentInformation.setSenderInformation(sender);
paymentInformation.setAdditionalInformation(additional);
PaymentManager.getInstance().initialize(ShopActivity.this, paymentInformation);
```

Finalizing payment process

A correctly finalized payment process is signalized by calling out `onPaymentSuccess` method, while payment error (both in parameters initialization and in latter stage) is signalized by calling out `onPaymentFailure` method. These events have the type `PaymentEndedEventArgs` argument with the following details:

PARAMETER	DESCRIPTION
Result	Type: PaymentResult Information on payment (amount, currency, control, channel id). A payment status is located in a field StateType. This parameter has an empty value, if transaction finishes with an error.
ErrorResult	Type: ProcessResult Describes the reason of finishing a payment process. A status different than "OK" means error. An announcement about payment problem should be displayed for a client.

Attention!!! A finalized payment process with a success doesn't mean the payment was processed, but it means the payment had no errors. A payment result will be returned in an appropriate event parameter.

Examples of event handlers:

```
private PaymentManagerCallback paymentManagerCallback = new PaymentManagerCallback() {

@Override
public void onPaymentSuccess(PaymentEndedEventArgs paymentEndedEventArgs) {

 if(paymentEndedEventArgs.getPaymentResult().getStateType()== StateType.COMPLETED) {

        // payment successful

}else if(paymentEndedEventArgs.getPaymentResult().getStateType()==
StateType.REJECTED) {

        // payment rejected

} else {

        // payment in progress

  }
}

@Override
public void onPaymentFailure(PaymentEndedEventArgs paymentEndedEventArgs) {

       // internal error during payment process,
   }
};
```

Summary details

On summary page it is possible to enable additional details like description, status and amount. Disabled by default.

In order to enable this functionality call method:

```
Configuration.setPaymentDetailsResultEnable(true);
```

Available currencies

A list of currencies supported by Dotpay can be downloaded by the following method of PaymentManager:

```
PaymentManager.getInstance().getCurrencies()
```

Changing presentation style

In order to change presentation style of controls elements in a payment process, proper setters of Configuration singleton have to be set. Settings have to be executed before initializing PaymentManager parameters.

Global settings:

PARAMETER	DESCRIPTION
ToolBarBackgroundColor	Type: int Defines toolbar background color
ToolBarTitleTextColor	Type: int Defines toolbar text color
ButtonTitleTextColor	Type: int Defines button text color
ButtonBackgroundColorResource	Type: int Defines button background color

Example:

```
Configuration.setToolBarBackgroundColor(R.color.red);
Configuration.setToolBarTitleTextColor(R.color.white);
Configuration.setButtonTitleTextColor(R.color.white);
Configuration.setButtonBackgroundColorResource(R.drawable.colorfulBtn);
```

Channels selecting control:

PARAMETER	DESCRIPTION
ChannelBackgroundColor	Type: int Defines view background color
ChannelBackgroundItemColor	Type: int Defines a single tile background color
ChannelBackgroundPressItemColor	Type: int Defines tile background color when pressed
ChannelItemTextColor	Type: int Defines tile text color
ChannelTextGravity	Type: int Defines tile text placement

Example:

```
Configuration.setChannelBackgroundColor(R.color.white);
Configuration.setChannelBackgroundItemColor(R.color.gray);
Configuration.setChannelBackgroundPressItemColor(R.color.green);
Configuration.setChannelTextGravity(Gravity.CENTER_HORIZONTAL);
Configuration.setChannelItemTextColor(R.color.black);
```

Favorite payment channel controller:

PARAMETER	DESCRIPTION
FavoriteChannelBackgroundItemColor	Type: int Defines a single tile background color
FavoriteChannelBackgroundPressItemColor	Type: int Defines tile background color when pressed

FavoriteChannelItemTextStyle	Type: int Defines tile text style

Example:

```
Configuration.setFavoriteChannelBackgroundItemColor(R.color.white);
Configuration.setFavoriteChannelBackgroundPressItemColor(R.color.gray);
```

Payment method change controller:

PARAMETER	DESCRIPTION
PaymentFormChannelText	Type: int Defines text in channel change controller
PaymentFormChannelTextColor	Type: int Defines text color in controller
PaymentFormChannelTextSize	Type: int Defines text size in controller
PaymentFormChannelTextGravity	Type: int Defines text placement in controller
PaymentFormChannelTextAllCaps	Type: boolean Sets letter to capital
PaymentAmountTextColor	Type: int Defines text color for amount
PaymentReceiverTextColor	Type: int Defines text color for receiver
PaymentDescriptionTextColor	Type: int

	Defines text color for description
PaymentInfoBackgroundColor	Type: int Defines payment information background color

Example:

```
Configuration.setPaymentFormChannelText(R.string.change_channel);
Configuration.setPaymentFormChannelTextColor(R.color.green);
Configuration.setPaymentFormChannelTextGravity(Gravity.RIGHT);
Configuration.setPaymentFormChannelTextSize(20);
Configuration.setPaymentFormChannelTextAllCaps(true);
Configuration.setPaymentInfoBackgroundColor(R.color.dpsdk_green);
```

It is also possible to change default status text colors. To change status color call method setStatusColor:

```
Configuration.setStatusColor(StateType.COMPLETED, R.color.green);
Configuration.setStatusColor(StateType.NEW, R.color.gray);
Configuration.setStatusColor(StateType.PROCESSING, R.color.gray);
Configuration.setStatusColor(StateType.REJECTED, R.color.red);
Configuration.setStatusColor(StateType.PROCESSING_REALISATION, R.color.gray);
Configuration.setStatusColor(StateType.PROCESSING_REALISATION_WAITING, R.color.gray);
```

Personal channel selection control

SDK library enables replacing default channel selection control, to adjust needs of Merchant's shop. The following steps are required to use this possibility:

1. In `PaymentManager` register personal channel selection activity (in the example it is `CustomChannelList`) with current context:

```
PaymentManager.getInstance().registerCustomChannelComponent((MenuActivity.this,
CustomChannelList.class);
```

2. Initialize payment in `PaymentManager` (in accordance with aforementioned chapter *Payment process initialization*)

3. Download channel list using one of the following methods:

   `PaymentManager.getInstance().getChannels()` – returns all channels

   `PaymentManager.getInstance().getChannels(isOnline)` – returns online/offline channels

   `PaymentManager.getInstance().getChannels(ids)` – returns channels according to ID

   `PaymentManager.getInstance().getChannels(paymentTypes)` – returns channels according to types

4. Set returned channel list as a data source for personal channel selection control.
5. After selecting channel by a client an appropriate PaymentManager method should be set:

   `PaymentManager.getInstance().initialPaymentForm(context, selectedChannel)`

Custom order summary controller

Library allows to change change order summarcy controller with your own, better suited for shop's needs. In order to use it:

1. Call method `setPaymentResultEnabled(boolean paymentResultEnabled)` singleton `Configuration` passing false as a value. It will disable order summary page from SDK.
2. The end of payment process is indicated by calling method `onPaymentSuccess` from interface `PaymentManagerCallback` passing `PaymentResult object.`
3. To request status from server call method

`getTransactionStatus(String id, String token, String number, String language)` singleton `PaymentManager`, where arguments can be downloaded from object `paymentEndedEventArgs.getResult();`

Example:

```
@Override

public void onPaymentSuccess(PaymentEndedEventArgs paymentEndedEventArgs){
    final PaymentResult paymentResult = paymentEndedEventArgs.getResult();
    new Thread(new Runnable(){
        @Override
        public void run(){
            try{

PaymentResultresult=PaymentManager.getInstance().getTransactionStatus(paymentResult.ge
tRecipientId(),paymentResult.getToken(),
paymentResult.getNumber(),paymentResult.getPaymentLanguage());

                // your code...

            }catch(OperationException e){

                // your code...

            }catch(NoConnectionException e){

                // your code...

            }
        }
    }).start();
}
```

Special channel support

In this chapter extra functions related to special payment channels are described.

Credit card payment - 1Click

1Click functionality enables quick payment process with a saved credit card. Basic credit card data are saved in Dotpay system.

This functionality (if available in Merchant's shop) is turned on by default in SDK. A client's consent is also required to use 1Click service (while filling out the payment form).

In order to turn off 1Click, the following command is required:

```
PaymentManager.getInstance().setOneClickEnabled(false);
```

ATTENTION

After turning off aforementioned options, formerly saved credit card data is not removed. To remove that data, the following commands are required.

Initializing 1Click payment

Payment process should be initialized with method `oneClickPayment PaymentManager`. Required arguments have been described in section ***Błąd! Nie można odnaleźć źródła odwołania..***

For development purposes method has special exceptions, which should be handled. They are listed in table below:

PARAMETER	DESCRIPTION
OneClickUnableException	Exception informs about disabled 1Click functionality
NotFoundPaymentCardException	Exception informs about no registered cards
NotFoundDefaultPaymentCardException	Exception informs about no default card Setting default card has been described in another section

Example:

```
String description = "Order no. 12345";
double amount = 123.45;

PaymentInformation paymentInformation = new PaymentInformation(merchant_Id, amount,
description, selectedCurrency);

Map<String, String> sender = new Map<String, String> {{"firstname", "Jan"},
{"lastname", "Kowalski"}, {"email", "jan.kowalski@test.pl"}}
Map<String, String> additional = new Map<String, String> {{"id1", "12345"},
{"amount1", "100"}, {"id2", "67890"}, {"amount2", "23.45"}}

paymentInformation.setSenderInformation(sender);
paymentInformation.setAdditionalInformation(additional);

try {
            PaymentManager.getInstance().oneClickPayment(this, paymentInformation);
        } catch (NotFoundPaymentCardException e) {

            // your code...

        } catch (NotFoundDefaultPaymentCardException e) {

             // your code...

        } catch (OneClickUnableException e) {

            // your code...

        }
}
```

Using built-in card management controller

In manager all remembered cards are listed. Manager also allows to add new cards.

ATTENTION: if functionality is disabled button will not be displayed.

In order to use built-in controller managing cards:

1. In xml layout, prepare controller FrameLayout.
2. Create instance *PaymentCardManagerFragment* via static method *newInstance* which should be passed to FragmentManager. *newInstance* method accepts arguments from table below:

PARAMETER	DESCRIPTION
merchant_id	Type: string Account ID for which payment is made
currency	Type: string currency of payment Downlaoding available currencies list has been described in section *Available curriencies*.
language	Type: string payment language

Example:

```
private void initPaymentCardManagerFragment() {

        FragmentManager fm = getSupportFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();
        Fragment fragment = PaymentCardManagerFragment.newInstance(merchant_id,
currency, language);
        ft.replace(R.id.fragment_container, fragment, null).commit();

        }
```

Card registration regulations

In order to register new card user has to accept two regulations. There are two methods which allow to change shop's name and redirect to shop's regulation available at given URL. To do this call `Configuration` singleton setters.

Available methods:

PARAMETER	DESCRIPTION
MerchantName	Type: String Defines shop's name
MerchantPolicyUrl	Type: String Redirects customer to shop's regulation available at given URL

Changing presentation style

To change presentation of card management controller elements call `Configuration` singleton setters. Configuration should be done before initiating layout containing card manager controller.

Card list controller:

PARAMETER	DESCRIPTION
PaymentCardManagerBackgroundColor	Type: int Defines view background color
PaymentCardManagerBackgroundItemColor	Type: int Defines a single tile background color
PaymentCardManagerBackgroundPressItemColor	Type: int Defines tile background controller when pressed
PaymentCardManagerTextStyle	Type: int Defines tile text style
PaymentCardManagerDefaultMarkColor	Type: int Defines marked icon color (available for API 17+)

Card adding form controller:

PARAMETER	DESCRIPTION
PaymentCardManagerFormBackgroundColor	Type: int Defines view background color
PaymentCardManagerFormLabelStyle	Type: int Defines controller label text style

Example:

```
Configuration.setPaymentCardManagerBackgroundColor(R.color.gray);
Configuration.setPaymentCardManagerTextStyle(R.style.CardManagerTextStyle);
Configuration.setPaymentCardManagerDefaultMarkColor(R.color.red);
Configuration.setPaymentCardManagerFormBackgroundColor(R.color.gray);
Configuration.setPaymentCardManagerFormLabelStyle(R.style.CardManagerLabelStyle);
```

Custom card management controller

To have data presented in a way better for shop, you can create own controller for card management downloading data from library.

1. Registered cards list

To download registered cards list call method:

`PaymentManager.getInstance().getPaymentCardList()` – method returns list of objects *PaymentCardInfo,* representing credit card.

2. Card registration

To register new card call method:

```
PaymentManager.getInstance().registerPaymentCard()
```

which accepts arguments from table below:

PARAMETER	DESCRIPTION
context	Type: Context Method call context
merchantId	Type: string Dotpay account ID
email	Type: string Customer email for registration
paymentCardData	Type: PaymentCardData Object containing card data
cardRegisteredCallback	Type: CardRegisteredCallback Callback awaiting the end of registration signal. Correctly registered card returns object `PaymentCardInfo`

3. Deleting card

To delete card call method:

`PaymentManager.getInstance().unregisterCardData(paymentCardId)` – which argument is available in object *`PaymentCardInfo.getPaymentCardId()`*. Lits of exceptions for this method has been described below:

PARAMETER	DESCRIPTION
PaymentOperationException	Exception sent by server containing event description.
NoConnectionException	Exception indicates network problems.

4. Setting default card

To make 1Click payment you have to set default card with method:

`PaymentManager.getInstance().setDefaultPaymentCard(paymentCardId)` – which argument is available in object *`PaymentCardInfo.getPaymentCardId()`*.

To check which card is currently marked as default call method:

`PaymentManager.getInstance().getDefaultPaymentCard()` – which returns object *`PaymentCardInfo`*.

Exceptions for this method are listed below:

PARAMETER	DESCRIPTION
NotFoundDefaultPaymentCardException	Exception indicates there is no default card.

Example:

```
PaymentManager.getInstance().registerPaymentCard(ShopActivity.this, merchantId, email,
paymentCardData, new CardRegisteredCallback() {
            @Override
            public void onSuccess(final PaymentCardInfo paymentCardInfo) {

PaymentManager.getInstance().setDefaultPaymentCard(paymentCardInfo.getCredit_card_id()
);
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        try {

PaymentManager.getInstance().unregisterCardData(paymentCardInfo.getCredit_card_id());
                        } catch (PaymentOperationException e) {

                            // your code ...

                        } catch (NoConnectionException e) {

                            // your code ...

                        }
                    }
                }).start();
            }
            @Override
            public void onFailure(ErrorCode errorCode) {

                            // your code ...

            }
        });
```

Transaction history and status

SDK library offers saving and displaying transaction history. The transaction history also displays related operations, e.g. subsequently made refunds and other additional operations.

The use of built-in control

In order to use built-in history control the following step is required:

1. Put in history control in xml file of selected layout. This layout will be responsible for displaying history, e.g.:

```
<fragment

android:name="pl.mobiltek.paymentsmobile.dotpay.fragment.TransactionHistoryFrag
ment"
android:id="@+id/transactionHistory"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

Changing presentation style

In order to change the way how control history elements are presented, proper `Configuration` singleton setters have to be set. Settings have to be executed before layout initialization that contains history control.

Channel selecting control:

PARAMETER	DESCRIPTION
HistoryTitleVisibility	Type: boolean Hide title
HistoryTitleText	Type: int Defines title text
HistoryTitleStyle	Type: int Defines title TextView style
HistoryDividerColor	Type: int Defines separator color under divider

HistoryBackgroundColor	Type: int Defines background color
setHistoryDateTextStyle	Type: int Defines TextView style for element data
setHistoryAmountTextStyle	Type: int Defines TextView style for element amount
HistoryDescriptionTextStyle	Type: int Defines TextView style for element description
HistoryBackgroundItemColor	Type: int Defines background color for whole element
HistoryDetailsHeaderTitleTextStyle	Type: int Defines TextView style for summary window title
HistoryDetailDividerColor	Type: int Defines TextView style for summary window title divider
HistoryDetailsTitleTextStyle	Type: int Defines TextView subtitle details style
HistoryDetailsValueTextStyle	Type: int Defines TextView details value style

Example:

```
Configuration.setHistoryTitleVisibility(true);
Configuration.setHistoryTitleText(R.string.HistoryTitle);
Configuration.setHistoryTitleStyle(R.style.HistoryTitleStyl);
Configuration.setHistoryDividerColor(R.color.black);
Configuration.setHistoryBackgroundColor(R.color.gray);
Configuration.setHistoryDateTextStyle(R.style.HistoryDateStyle);
Configuration.setHistoryAmountTextStyle(R.style.HistoryAmountStyle);
Configuration.setHistoryDescriptionTextStyle(R.style.HistoryDescriptionStyle);
Configuration.setHistoryBackgroundItemColor(R.color.white);
Configuration.setHistoryDetailsHeaderTitleTextStyle(R.style.HistoryTitleStyle);
Configuration.setHistoryDetailDividerColor(R.color.black);
Configuration.setHistoryDetailsTitleTextStyle(R.style.HistoryDetailsTitleStyle);
Configuration.setHistoryDetailsValueTextStyle(R.style.HistoryDetailsValueStyle);
```

Personal history control

SDK library offers creating personal history control to better match web shop specification.

In order to download information on transaction list, the following PaymentManager method is required:

```
PaymentManager.getInstance().loadTransactios();
```

Additionally, for transactions presented in personal history you can:

1. Remove a single record:

```
PaymentManager.getInstance().deleteTransaction(paymentResult);
```

2. Check current transaction status:

```
PaymentManager.getInstance().getTransactionStatus(paymentResult);
```

List of exceptions for this method has been described below:

PARAMETER	DESCRIPTION
PaymentOperationException	Exception from server containing even description.
NoConnectionException	Exception indicates network problems.