TrustPay API

Version 1.7

2013-06-21

Terminology

"Client Application" - The application installed on the handset that wishes to make use of the TrustPay billing services.

"TrustPay API" - A library that is linked to the Client Application and which provided facilities to interact with the TrustPay Application

"TrustPay Application" - The application installed on the handset to mediate between the client application and the TrustPay backend.

"TrustPay Backend" - The server infrastructure taking care of operations, billing and administration around the TrustPay service.

Theory of Operation

The TrustPay application acts as an intermediary between the client application and the consumer who wants to pay for something.

This allows the client app to be installed with very few permissions, increasing the likelyhood that a consumer will install it.

When the client app wishes to bill the consumer, the request is passed on to the TrustPay app. This app then presents a screen with all available billing methods for the amount. The consumer selects a method. A method-specific screen gets displayed to the consumer to supply information and/or to validate the request. If the consumer accepts, the billing takes place and the result is returned to the client application.

Android API

There are three separate mechanisms for communicating with the Android TrustPay app and backend, depending on the purpose of the communication.

Firstly, a bound service is used to interrogate it to find out what options are available, without requiring consumer interaction.

Secondly, an Intent is launched for effect, to allow the consumer to interact with the TrustPay app to actually bill.

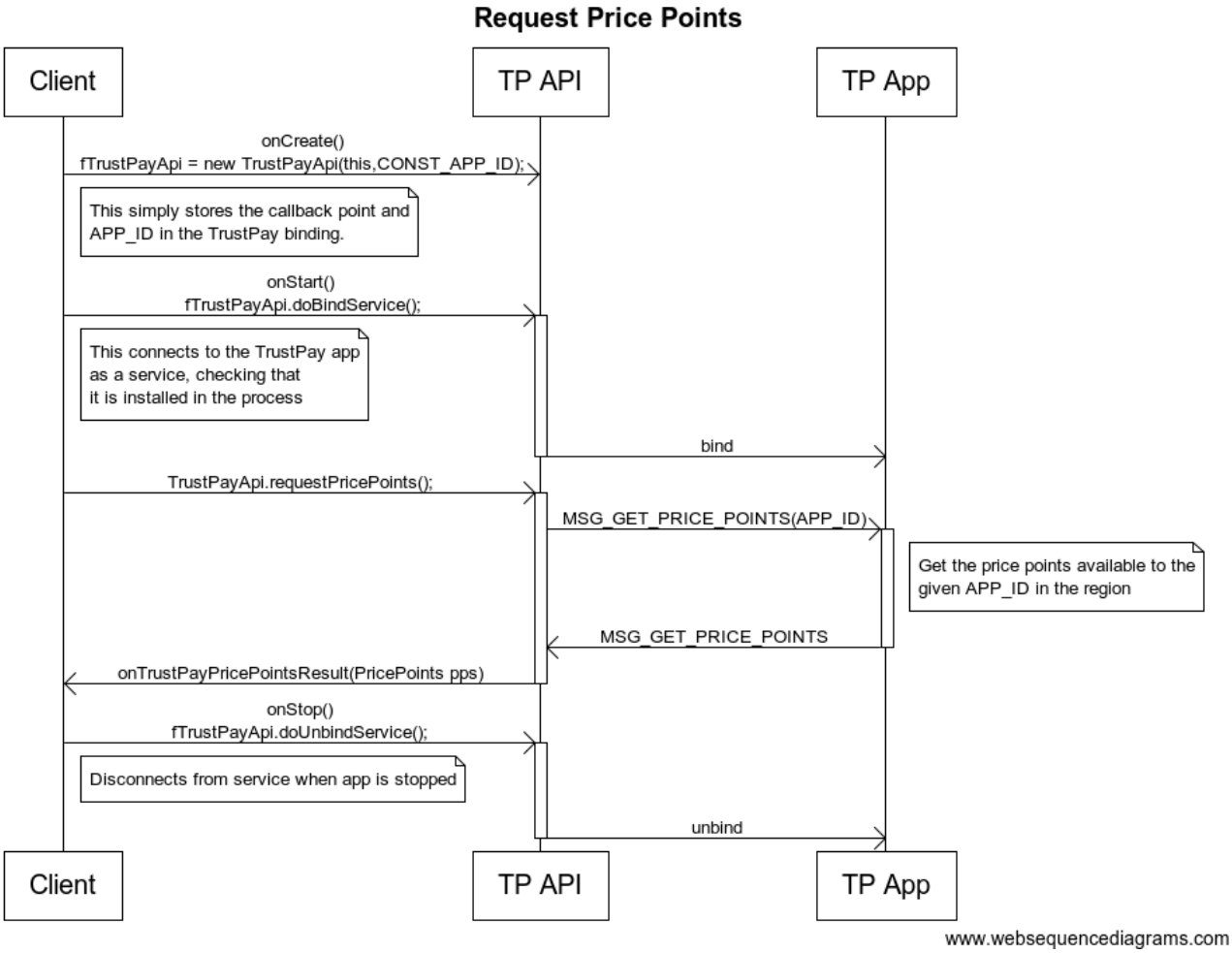
Thirdly, when the TrustPayApi service is bound, the TrustPay Application wil automatically refresh all cached billing methods in the background if Auto-Refresh is enabled. If not, the api can be instructed by the developer to refresh.

The overall process in the client app is managed in an Activity, which provides an entry point for the callbacks from the TrustPay app. The typical scenarios are presented below.

If the Client Application cannot bind to the TrustPay Application, the chance is that the TrustPay Application has not been installed yet. The API exposes a method to Download the appropriate TrustPay Application in this case.

Requesting Price Points

This function is used for the Client App to get the price points supported by TrustPay for the current handset territory and network.



In this case it is important to override the onStart and onStop Activity methods to bind and unbind from the TrustPay application.

Code fragments from ApiDemo

```
public class ApiDemo extends Activity implements PricePointListener{
/**The class must extend a Object that extends Context and implement PricePointListener*/

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        …

        fTrustPayApi = new TrustPayApi(this,CONST_APP_ID);
```

```
    fTrustPayApi.setPricePointListener(this);
}

@Override
protected void onStart() {
    super.onStart();
    fTrustPayApi.doBindService(true);
    // Pass in true for Auto-Refresh of pricepoints, false to disable
}

@Override
protected void onStop() {
    super.onStop();
    fTrustPayApi.doUnbindService();
}

public boolean getPricePoints(boolean isTest) {

    fTrustPayApi.setTest(isTest);
    return fTrustPayApi.requestPricePoints();

}

// If the TrustPay application cannot be found, getPricePoints will return false
public void …….

    if(!getPricePoints(test)){
            // Could not bind to TrustPayApi,might not be installed, try to download
            TrustPayApi.downloadTrustPay(this);
    }


@Override
public void onTrustPayPricePointsResult(PricePoints pps) {
    if (pps != null) {
        // Do something with the returned PricePoints structure
    }
}
```

The returned PricePoints object has the following format

```
    public class PricePoints implements Serializable {
            public ArrayList<PricePoint> fPricePoints;
    ….
    }
```

Each PricePoint has

```
            String fCurrency
            String fLower
            String fUpper
            String fStep
            String fType
```

where: fCurrency is always populated with the currency code :
e.g. fCurrency = “ZAR” for South-African rands

fLower is the lower value of the pricepoint range
fUpper is the upper value of the pricepoint range
fStep is the increment values between lower and upper
e.g.: for all full rand values between 20 and 99, the values will be:
fLower = "20.00" fUpper = "99.00" fStep = "1.00"
for all cent values between 1 and 100 rand, the values will be:
fLower = "1.00" fUpper ="100.00" fStep = "0.01"
for a single pricepoint of 5 rand, the values will be:
fLower = "5.00" fUpper = "5.00" fStep = "0.00"
fType is the transaction type, values currently supported are:
CARRIER BILLING
VOUCHER
CREDIT CARD
WALLET

Request Payment

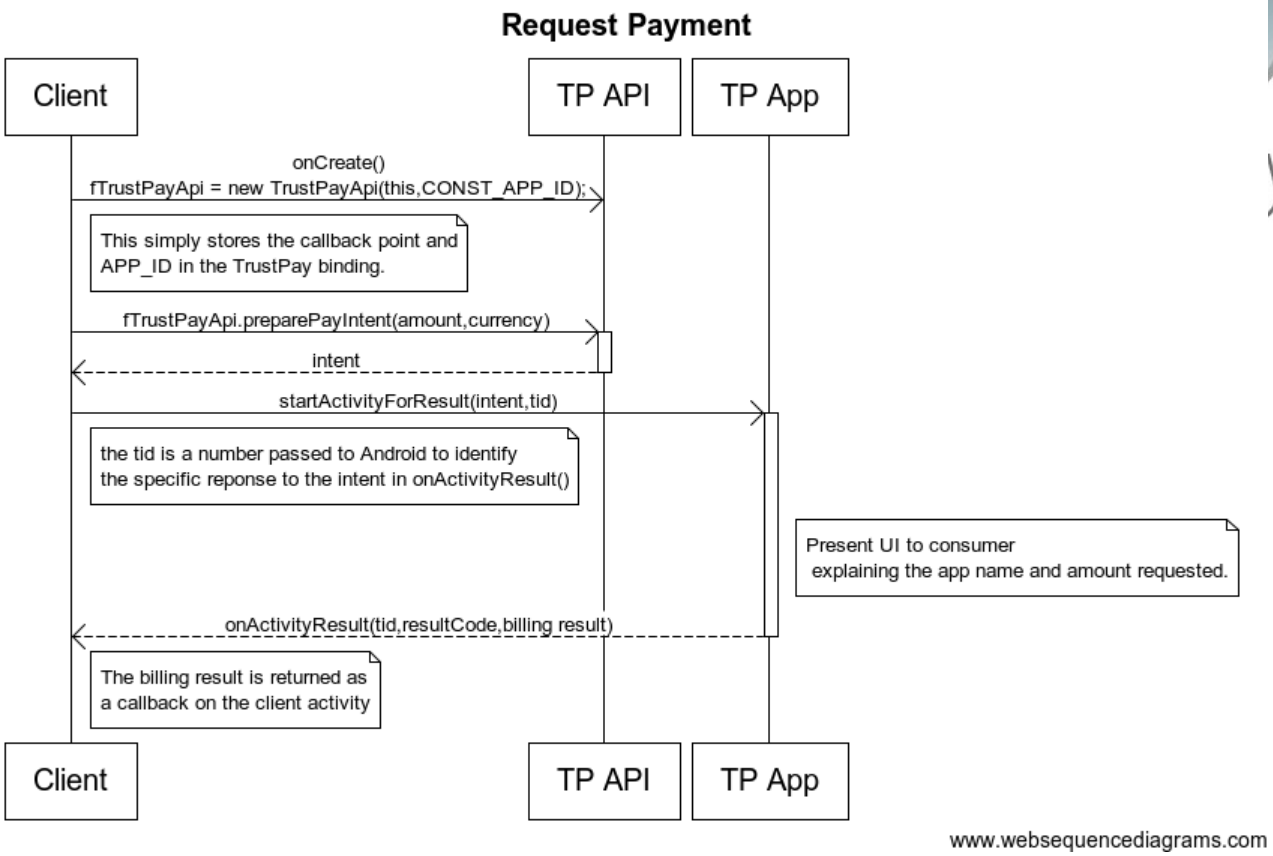
This function is used to actually bill the consumer. It does this by presenting a screen explaining the amount to be billed and the name of the application requesting it. The consumer can either agree to bill, or cancel the transaction. The status is returned to the calling app.

Code fragments from ApiDemo for Payment

```
public class ApiDemo extends Activity implements PricePointListener{
/**The class must extend a Object that extends Context and implement PricePointListener*/

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        …

        fTrustPayApi = new TrustPayApi(this,CONST_APP_ID);
        fTrustPayApi.setPricePointListener(this);
    }
```



/* The third parameter in the fTrustPayApi.preparePayIntent call is the payment methods that you want to support the [*] wildcard means 'All methods supported for my transaction'
If the pricepoints returned indicated that you can do Credit Card, Voucher and Carrier billing, and you only want to support Credit Card and Voucher, you pass the parameter as: [CREDIT CARD;VOUCHER] and only those two options will be displayed to the user. (See PricePoint description earlier for Payment Method codes.
app_user, is a paramater that will identify the specific mobile user,
tx_Description is a human-readble decription of the transaction for reporting purposes,
app_reference is a transaction number assigned by the developer.*/

```
        public void pay(String amount,boolean isTest) {
          fTrustPayApi.setTest(isTest);
              Intent intent = fTrustPayApi.preparePayIntent(amount,"ZAR",”[*]”,app_user,
tx_Description, app_reference);

              try {
                    startActivityForResult(intent,123);
              } catch (ActivityNotFoundException e) {
                    TrustPayApi.downloadTrustPay(this);
              }
        }

       // This method is called when the billing is done
   @Override
   protected void onActivityResult(int requestCode, int resultCode, Intent data) {

       EditText t = (EditText) findViewById(R.id.editText1);
      if (resultCode == Activity.RESULT_OK && requestCode == 123) {
       BillingResult res = (BillingResult) data.getSerializableExtra("result");
       t.setText("got result:"+res);
```

```
    } else if (resultCode == Activity.RESULT_CANCELED && requestCode == 123) {
      BillingResult res = (BillingResult) data.getSerializableExtra("result");
      t.setText("got result:"+res);
    } else {
      String res = "got strange result: (resultCode,requestCode):(" + resultCode +
              "," + requestCode + ")";
      t.setText(res);
    }
  }
```

The BillingResult object has the following format

```
public class BillingResult implements Serializable {

    // Request values

    public CharSequence fAppid     = null;
    public CharSequence fOperation = null;
    public CharSequence fAmount    = null;
    public CharSequence fCurrency  = null;
    public CharSequence fAppref    = null;
    public Boolean      fIsTest    = null;

    // Returned values

    public CharSequence fTrustPayRef = null;
    public CharSequence fMessage     = null;
    public boolean      fSuccess     = false;
....
}
```

The important field is fSuccess, which will be true if the billing succeeded.
FMessage provides more information on the kind of problem that occurred. Possible values are listed in the table below.

Value	**Interpretation**
INVALID_OPERATION	The billing intent was delivered with an invalid operation. If the API is used, this should never occur.
Test success	A successful simulated billing occurred. In other words, the consumer clicked on accept when asked to pay, but no payment was actually made.
Cancelled by user	The consumer chose not to pay
Success	Successful billing occurred. In other words, the consumer clicked on accept when asked to pay, and payment was made.
Error	A generic error occurred that prevented billing
Error: No service	The handset is currently not connected to a mobile network

Error: Radio off	The handset is currently not connected to a mobile network, because the radio is off in e.g. Flight Mode

Refresh Payment Methods

This function is used to actually refresh the Payment Methods and Price Points on the device by doing a delta synchronization with the TrustPay backend environment.

Code fragments from ApiDemo

```
public class ApiDemo extends Activity implements PricePointListener{
/**The class must extend a Object that extends Context and implement PricePointListener*/

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        …

        fTrustPayApi = new TrustPayApi(this,CONST_APP_ID);
        fTrustPayApi.setPricePointListener(this);
    }

    @Override
    protected void onStart() {
        super.onStart();
        fTrustPayApi.doBindService(true);
        // Pass in true for Auto-Refresh of pricepoints, false to disable
    }
```

…..
After fTrustPayApi.doBindService has been called, the Refresh Payment Methods can be done by calling :

```
fTrustPayApi.requestUpdate();
```

When done, the result gets returned in

```
@Override
public void onUpdateComplete(Boolean arg0) {

}
```

where arg0 is true when successful or false on a failure.

Changes

Date	Version	Changes
2013-06-21	1.7	New Transaction Types
2013-01-04	1.6	Added App_reference to the API
2012-12-14	1.5	Added Caching of PricePoints and refreshing thereof.
2012-10-24	1.4	Implemented Payment Method selection, Appuser Transaction Description params in the preparePayIntent method.
2012-10-08	1.3	Developers can now Implement PricePointListener instead of extending and activity.
2012-08-24	1.2	New pricepoint format supporting all payment methods. Add Activity.*RESULT_CANCELED* as a Activity return resultCode.
2012-04-25	1.1	Added fValueNett field to the PricePoints structure, to give an indication of nett payout for the pricepoint
2012-04-10	1.0	Initial Version