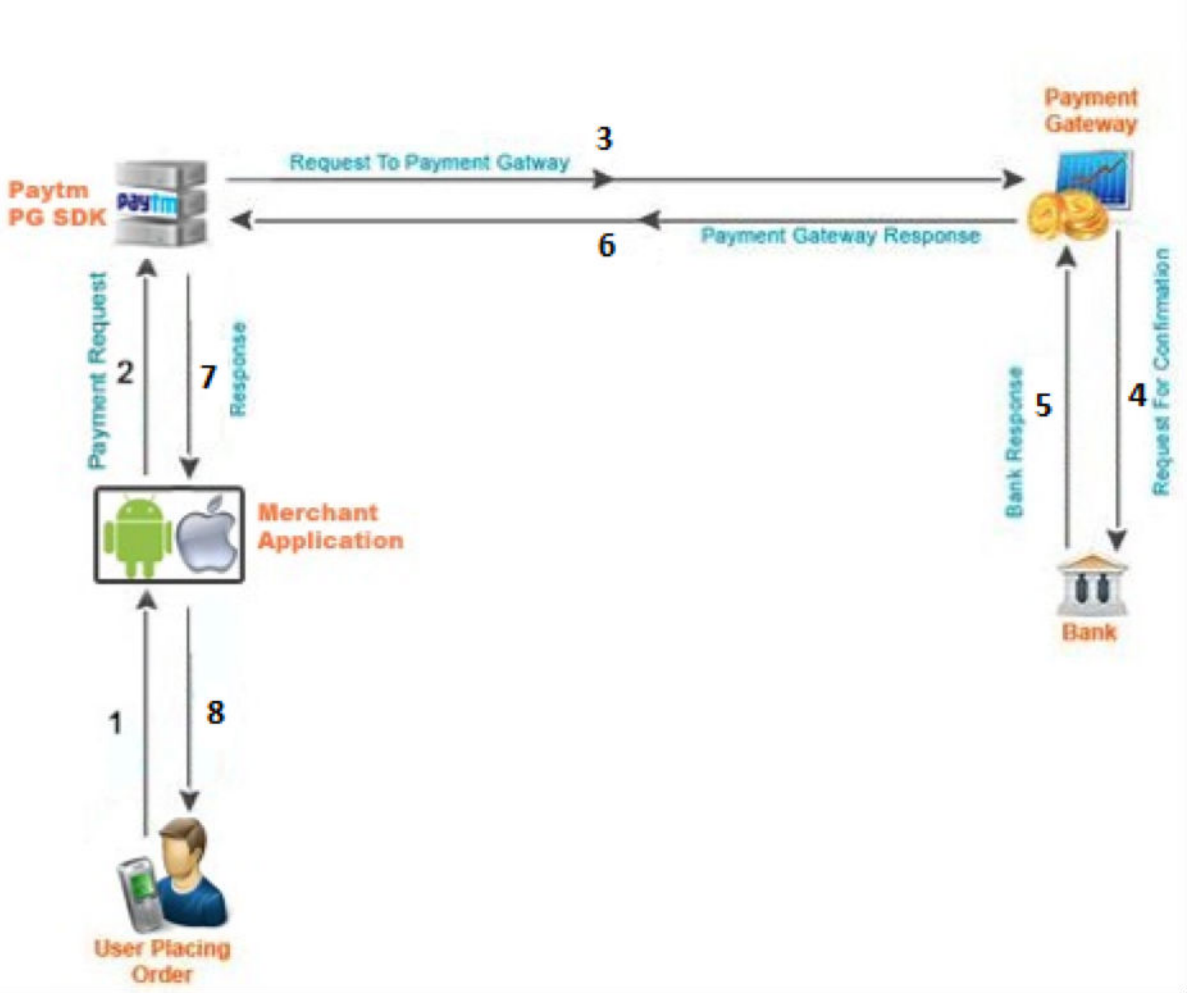
Paytm SDK Integration Guide for Mobile Applications

Introduction

The Paytm Payments SDK for iOS and Android enables merchants to integrate payment gateway into their mobile apps. The following diagram describes the flow at a broad level.



Flow

1. Merchant creates the Paytm payload and checksum on server side. Checksum must be created as per Paytm checksum logic. For checksum logic, kits are available on Github in different languages.
2. Merchant app invokes Paytm SDK with the payload (created on server side in 1st step).
3. SDK posts all the parameters to Paytm server by redirecting the user in a webview.
4. On Paytm cashier page, customer selects a payment option and submits details.
5. Paytm redirects customer to bank page where he/she completes the payment on bank end.
6. Once the payment is completed on bank page, bank redirects the customer back to Paytm.
7. Paytm verifies the response from bank and sends response back to SDK.
8. Merchant app gets the response from SDK and moves the data to server for verification of checksum, amount and status.
9. Before marking the order status as success merchant should re-verify the transaction status and amount by calling Paytm status query API from their server.
10. Merchant App then displays the payment/order status to the customer accordingly.

NOTE:

To prevent missuse and fraud, every merchant has to implement following security recommendations:

1. **Checksum is an essential mechanism to ensure that no tampering occurs during data exchange over the network. It is merchant responsibility to ensure the confidentiality of merchant key and checksum logic, so kindly make sure no third party can access them.**

2. **As per secure practices, before marking transaction as success in system after receiving success response from Paytm SDK, merchant should verify transaction status and amount of order again with Paytm by calling status check API from their back-end server. Paytm transaction status API returns the ORDERID, TXNAMOUNT and STATUS as per record in Paytm system for a given order. If ORDERID and TXNAMOUNT parameters are not matching with the order data on merchant side or STATUS is not TXN_SUCCESS, merchant should not fulfill/deliver this order.**

Request Payload Parameters (with sample)

1. **MID** : PAYTM_MERCHANT_ID **// Merchant Identifier provided By Paytm**
2. **ORDER_ID :** ORDER0000000001 **// Unique Order ID for every request**
3. **CUST_ID :** 10000988111 **// Unique customer Identifier**
4. **INDUSTRY_TYPE_ID :** PAYTM_INDUSTRY_TYPE_ID **// Industry provided By Paytm**
5. **CHANNEL_ID :** WAP **// channel provided By Paytm**
6. **TXN_AMOUNT :** 1 **// Transaction Amount**
7. **WEBSITE :** PAYTM_WEBSITE **// website parameter provided By Paytm**
8. **CALLBACK_URL :** https://www.paytm.com **// provided By Paytm**
9. **EMAIL :** abc@gmail.com **// customer Email Id**
10. **MOBILE_NO :** 9999999999 **// 10 digit customer mobile no**
11. **CHECKSUMHASH :** "108 character string" **//checksum hash created using Paytm utility**

Response Payload Parameters (with sample)

1. **MID** : PAYTM_MERCHANT_ID **// Merchant Identifier provided By Paytm**
2. **ORDERID :** ORDER0000000001 **// Unique Order ID sent by merchant in request**
3. **TXNID :** 9829892841900981 **// Unique Transaction Id sent by Paytm**
4. **TXNAMOUNT :** 1.00 **// Transaction Amount**
5. **PAYMENTMODE :** CC **// Mode By which payment is completed**
6. **CURRENCY :** INR **// payment currency**
7. **TXNDATE :** 2017-04-02 02:16:07.0 **// transaction date and time**
8. **STATUS :** TXN_SUCCESS **// transaction status**
9. **RESPCODE :** 01 **// Transaction response code**
10. **RESPMSG : T**xn Success **// Transaction response message**
11. **GATEWAYNAME :** HDFC **// Gateway name**
12. **BANKTXNID :** 862229160270920 **// Bank Transaction Id**
13. **BANKNAME :** HDFC **// Bank Name**
14. **CHECKSUMHASH :** "108 character response checksum string" **//checksum hash created using Response parameter's merchant needs to verify this.**

Android:

Basic setup:

Update the ***AndroidManifest.xml*** file of your application as shown below.

- 2.1 Add the **INTERNET** and **ACCESS_NETWORK_STATE** permissions to your ***AndroidManifest.xml***, if it is not added already. These two permissions are required for PGSDK Service to run. Make sure that these two permissions are specified in your manifest file.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

- 2.2 Add the Declaration of ***PaytmPGActivity*** in your ***AndroidManifest.xml***. This activity is present in the PGSDK. Make sure that your manifest file also specify this activity element.

```
<activity
  android:name="com.paytm.pgsdk.PaytmPGActivity"
  android:screenOrientation="portrait"
  android:configChanges="keyboardHidden|orientation|keyboard"/>
```

Payment Transaction

1. Obtain the ***PaytmPGService*** instance. You can choose a staging or production instance as per need. ***PaytmPGService.getStagingService()*** will return the Service pointing to Staging Environment. ***PaytmPGService.getProductionService()*** will return the Service pointing to Production Environment.

```
PaytmPGService Service = PaytmPGService.getStagingService();
```

2. Create a HASHMAP Object holding the Order Information. Syntax for creating HASHMAP Object is as follows:

```
Map<String, String> paramMap = new HashMap<String,String>();

paramMap.put( "MID" , "PAYTM_MERCHANT_ID");
paramMap.put( "ORDER_ID" , "ORDER0000000001");
paramMap.put( "CUST_ID" , "10000988111");
paramMap.put( "INDUSTRY_TYPE_ID" , "PAYTM_INDUSTRY_TYPE_ID");
paramMap.put( "CHANNEL_ID" , "WAP");
paramMap.put( "TXN_AMOUNT" , "1");
paramMap.put( "WEBSITE" , "PAYTM_WEBSITE");
paramMap.put( "CALLBACK_URL" , "https://www.paytm.com");
paramMap.put( "EMAIL" , "abc@gmail.com");
paramMap.put( "MOBILE_NO" , "9999999999");
paramMap.put( "CHECKSUMHASH" ,
"w2QDRMgp1/BNdEnJEAPClOmNgQvsi+BhpqijfM9KvFfRiPmGSt3Ddzw+oTaGCLneJwx
FFq5mqTMwJXdQE2EzK4px2xruDqKZjHupz9yXev4=");
```

ORDER_ID: Merchant's Transaction ID which should be unique for every request.

CUST_ID: Customer identifier, which is a unique user Id that the Merchant has assigned to its customers. It should be unique for every new customer.

TXN_AMOUNT: Amount that is to be charged to the customer

EMAIL: Customer's Email Id.

MOBILE_NO: Customer's Mobile Number.

CHECKSUMHASH: You have to create checksum with all Paytm transactional parameter at your server side

3. (Optional) Create PaytmClientCertificate Object holding the Certificate information.

Syntax for creating PaytmClientCertificate Object is as follows

```
PaytmClientCertificate Certificate = new PaytmClientCertificate(String inPassword, String
inFileName);
```

inPassword: Client side certificate password.
inFileName: Client side certificate file name. This file must be present inside raw folder.

4. Now create the order object

```
PaytmOrder Order = new PaytmOrder(paramMap);
```

5. Call ***initialize()*** method to set ***PaytmOrder*** and ***PaytmClientCertificate*** objects.

   Method Signature is as follows

   ```
   Service.initialize(Order, null);
   ```

   PaytmOrder: Object which is holding Order Information.
   PaytmClientCertificate: Object which is holding Certificate Information or null.

6. Call ***startPaymentTransaction()*** method to start the Payment transaction.

   ```
   Service.startPaymentTransaction(this, true, true, new PaytmPaymentTransactionCallback()
   ```

   Method Signature is as follows

   a. Activity Conext, in which you are calling this method
   b. Boolean variable to show or hide header bar
   c. Boolean variable to send callback (always true)
   d. ***PaytmPaymentTransactionCallback*** Instance to send callback message.

7. Implement the callback methods to handle the response upon payment completion.

   Callback methods are under ***PaytmPaymentTransactionCallback***

   ```
   public void someUIErrorOccurred(String inErrorMessage) { /*If any UI Erroe Occur*/ }

   public void onTransactionResponse(Bundle inResponse) { /*once transaction is completed on paytm you get complete response in json format*/ }

   public void networkNotAvailable() {  /* If network is not available, then this method gets called. */ }

   public void clientAuthenticationFailed(String inErrorMessage) { /* This method gets called if client authentication  failed. // Failure may be due to following reasons 1. Server error or downtime.  Error Message describes the reason for failure.*/ }

   public void onErrorLoadingWebPage(int iniErrorCode, String inErrorMessage, String inFailingUrl) { }
   ```

```
public void onBackPressedCancelTransaction() {/*on back press event this callback got triggered */}

public void onTransactionCancel(String inErrorMessage, Bundle inResponse) {/*on transaction cancelled this callback got triggered */ }
```

Example:

```
//Getting the Service Instance. PaytmPGService.getStagingService() will return //the Service pointing to Staging
Environment.
//and PaytmPGService.getProductionService() will return the Service pointing to //Production Environment.

PaytmPGService Service = null;
Service = PaytmPGService.getStagingService();
                    or
Service = PaytmPGService.getProductionService();

//Create new order Object having all order information.
Map<String, String> paramMap = new HashMap<String,String>();
paramMap.put( "MID" , "PAYTM_MERCHANT_ID");
paramMap.put( "ORDER_ID" , "ORDER0000000001");
paramMap.put( "CUST_ID" , "10000988111");
paramMap.put( "INDUSTRY_TYPE_ID" , "PAYTM_INDUSTRY_TYPE_ID");
paramMap.put( "CHANNEL_ID" , "WAP");
paramMap.put( "TXN_AMOUNT" , "1");
paramMap.put( "WEBSITE" , "PAYTM_WEBSITE");
paramMap.put( "CALLBACK_URL" , "https://www.paytm.com");
paramMap.put( "EMAIL" , "abc@gmail.com");
paramMap.put( "MOBILE_NO" , "9999999999");
paramMap.put( "CHECKSUMHASH" ,
"w2QDRMgp1/BNdEnJEAPCIOmNgQvsi+BhpqijfM9KvFfRiPmGSt3Ddzw+oTaGCLneJwxFFq5mqT
MwJXdQE2EzK4px2xruDqKZjHupz9yXev4=");

//Create Client Certificate object holding the information related to Client Certificate. Filename
must be without .p12 extension.
//For example, if suppose client.p12 is stored in raw folder, then filename
must be client.
PaytmClientCertificate Certificate = new PaytmClientCertificate
("password" , "filename" );
```

```
//Set PaytmOrder and PaytmClientCertificate Object. Call this method
and set both objects before starting transaction.
Service.initialize(Order, Certificate);
//OR
Service.initialize(Order, null);


//Start the Payment Transaction. Before starting the transaction ensure that initialize method is called.

Service.startPaymentTransaction(this, true, true, new PaytmPaymentTransactionCallback() {

        @Override
        public void someUIErrorOccurred(String inErrorMessage) {
          Log.d("LOG", "UI Error Occur.");
                Toast.makeText(getApplicationContext(), " UI Error Occur. ",
Toast.LENGTH_LONG).show();
        }

        @Override
        public void onTransactionResponse(Bundle inResponse) {
                Log.d("LOG", "Payment Transaction : " + inResponse);
                Toast.makeText(getApplicationContext(), "Payment Transaction response
"+inResponse.toString(), Toast.LENGTH_LONG).show();
        }

        @Override
        public void networkNotAvailable() {
          Log.d("LOG", "UI Error Occur.");
                Toast.makeText(getApplicationContext(), " UI Error Occur. ",
Toast.LENGTH_LONG).show();
        }

        @Override
        public void clientAuthenticationFailed(String inErrorMessage) {
          Log.d("LOG", "UI Error Occur.");
                Toast.makeText(getApplicationContext(), " Severside Error "+ inErrorMessage,
Toast.LENGTH_LONG).show();
        }

        @Override
        public void onErrorLoadingWebPage(int iniErrorCode,
            String inErrorMessage, String inFailingUrl) {

        }
```

```
        @Override
        public void onBackPressedCancelTransaction() {
          // TODO Auto-generated method stub
        }

        @Override
        public void onTransactionCancel(String inErrorMessage, Bundle inResponse) {
         Log.d("LOG", "Payment Transaction Failed " + inErrorMessage);
         Toast.makeText(getBaseContext(), "Payment Transaction Failed ",
Toast.LENGTH_LONG).show();
       }

    });
```

IOS

1. Merchant Configuration
   SDK identifies each Merchant by **PGMerchantConfiguration** object. This is a singleton class and needs to be configured once.
   ```
   PGMerchantConfiguration *mc = [PGMerchantConfiguration defaultConfiguration];
   ```

2. Prepare an order for your customer. You can get a **PGOrder** object by calling the following class method with below parameters

   ```
   NSMutableDictionary *orderDict = [NSMutableDictionary new];
   //Merchant configuration in the order object
   orderDict[@"MID"] = @"PAYTM_MERCHANT_ID";
   orderDict[@"ORDER_ID"] = @"TestORDER000001";
   orderDict[@"CUST_ID"] = @"123456";
   orderDict[@"INDUSTRY_TYPE_ID"] = @"PAYTM_INDUSTRY_TYPE_ID";
   orderDict[@"CHANNEL_ID"] = @"WAP";
   orderDict[@"TXN_AMOUNT"] = @"1";
   orderDict[@"WEBSITE"] = @"PAYTM_WEBSITE";
   orderDict[@"CALLBACK_URL"] = @"https://paytm.paytm.com";
   orderDict[@"EMAIL"] = @"abc@gmail.com";
   orderDict[@"MOBILE_NO"] = @"9999999999";
   orderDict[@"CHECKSUMHASH"] =
@"PsMxOJ40gDWSg5KanxJ9rP62SwQE7PbAN0ybzxOX8qyu228jcCX4+U1Km3scoDFfvpSsSt7Nm1+XGA37taOdLm6/U8myEvvToiy0wKcfexs=";

   PGOrder *order = [PGOrder orderWithParams:orderDict];
   ```

   Parameters Descriptions:

   ORDER_ID: Merchant's Transaction ID which should be unique for every request.

   CUST_ID: Customer identifier, which is a unique user Id that the Merchant has assigned to its customers.

   TXN_AMOUNT: Amount that is to be charged to the customer

   EMAIL: Customer's Email Id.

   MOBILE_NO: Customer's Mobile Number.

   CHECKSUMHASH: checksumhash

3. Once your order is ready, You can start transaction by Initializing PGTransactionViewController with your order. Then push this with Navigation controller.

```
[PGServerEnvironment selectServerDialog:self.view completionHandler:^(ServerType
eServerTypeProduction)
{
PGTransactionViewController *txnController = [[PGTransactionViewController alloc]
initTransactionForOrder:order];

    txnController.serverType = eServerTypeProduction;
    txnController.merchant = mc;
    txnController.delegate = self;
    txnController.loggingEnabled = YES;
    [self showController:txnController];

    }];
```

To hide logs set loggingEnabled to NO

4. Implement The PGTransactionDelegate protocol.

```
//Once transaction is completed you get response from this delegate

-(void)didFinishedResponse:(PGTransactionViewController *)controller response:(NSString
*)responseString {
    DEBUGLOG(@"ViewController::didFinishedResponse:response = %@", responseString);
    NSString *title = [NSString stringWithFormat:@"Response"];
    [[[UIAlertView alloc] initWithTitle:title message:[responseString description] delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil] show];
    [self removeController:controller];
}
// on cancel the transaction this delegate is called

- (void)didCancelTransaction:(PGTransactionViewController *)controller error:(NSError*)error
response:(NSDictionary *)response
{
    DEBUGLOG(@"ViewController::didCancelTransaction error = %@ response= %@", error,
response);
    NSString *msg = nil;
    if (!error) msg = [NSString stringWithFormat:@"Successful"];
    else msg = [NSString stringWithFormat:@"UnSuccessful"];

    [[[UIAlertView alloc] initWithTitle:@"Transaction Cancel" message:msg delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil] show];
    [self removeController:controller];
```

```
}

- (void)didFinishCASTransaction:(PGTransactionViewController *)controller
response:(NSDictionary *)response
{
    DEBUGLOG(@"ViewController::didFinishCASTransaction:response = %@", response);
}
```

Example:

```
-(IBAction)testPayment:(id)sender
{
    //Step 1: Create a default merchant config object
    PGMerchantConfiguration *mc = [PGMerchantConfiguration defaultConfiguration];

    //Step 2: Create the order with whatever params you want to add. But make sure that you
include the merchant mandatory params
    NSMutableDictionary *orderDict = [NSMutableDictionary new];
    //Merchant configuration in the order object
orderDict[@"MID"] = @"PAYTM_MERCHANT_ID";
orderDict[@"ORDER_ID"] = @"TestORDER000001";
orderDict[@"CUST_ID"] = @"123456";
orderDict[@"INDUSTRY_TYPE_ID"] = @"PAYTM_INDUSTRY_TYPE_ID";
orderDict[@"CHANNEL_ID"] = @"WAP";
orderDict[@"TXN_AMOUNT"] = @"1";
orderDict[@"WEBSITE"] = @"PAYTM_WEBSITE";
orderDict[@"CALLBACK_URL"] = @"https://paytm.paytm.com";
orderDict[@"EMAIL"] = @"abc@gmail.com";
orderDict[@"MOBILE_NO"] = @"9999999999";
orderDict[@"CHECKSUMHASH"] =
@"PsMxOJ40gDWSg5KanxJ9rP62SwQE7PbAN0ybzxOX8qyu228jcCX4+U1Km3scoDFfvpSsSt7Nm
1+XGA37taOdLm6/U8myEvvToiy0wKcfexs=";


    PGOrder *order = [PGOrder orderWithParams:orderDict];

    //Step 3: Choose the PG server. In your production build dont call selectServerDialog. Just
create a instance of the
    //PGTransactionViewController and set the serverType to eServerTypeProduction
    [PGServerEnvironment selectServerDialog:self.view completionHandler:^(ServerType
eServerTypeProduction)
     {
        PGTransactionViewController *txnController = [[PGTransactionViewController alloc]
initTransactionForOrder:order];
         //txnController.merchant = [PGMerchantConfiguration defaultConfiguration];
```

```
        txnController.serverType = eServerTypeProduction;
        txnController.merchant = mc;
        txnController.delegate = self;
        txnController.loggingEnabled = YES;
        [self showController:txnController];
    }];
}

#pragma mark PGTransactionViewController delegate

-(void)didFinishedResponse:(PGTransactionViewController *)controller response:(NSString
*)responseString {
    DEBUGLOG(@"ViewController::didFinishedResponse:response = %@", responseString);
    NSString *title = [NSString stringWithFormat:@"Response"];
    [[[UIAlertView alloc] initWithTitle:title message:[responseString description] delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil] show];
    [self removeController:controller];
}

- (void)didCancelTransaction:(PGTransactionViewController *)controller error:(NSError*)error
response:(NSDictionary *)response
{
    DEBUGLOG(@"ViewController::didCancelTransaction error = %@ response= %@", error,
response);
    NSString *msg = nil;
    if (!error) msg = [NSString stringWithFormat:@"Successful"];
    else msg = [NSString stringWithFormat:@"UnSuccessful"];

    [[[UIAlertView alloc] initWithTitle:@"Transaction Cancel" message:msg delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil] show];
    [self removeController:controller];
}

- (void)didFinishCASTransaction:(PGTransactionViewController *)controller
response:(NSDictionary *)response
{
    DEBUGLOG(@"ViewController::didFinishCASTransaction:response = %@", response);
}
```

Checksumhash

- Checksum must include all the Paytm parameters (Mandatory/Optional) which will come in request.
- All the mandatory parameters must come and must contain value.
- If Merchant wants to send any optional parameter in the request (Depending on requirement) then he needs to include it in CHECKSUMHASH as well.
- Sometimes extra parameters come because of browser form submit or from framework then we will filter them out those parameters and will include only valid parameters in our CHECKSUMHASH.
- If any optional parameter is empty then don't include that parameter in your CHECKSUMHASH and even don't send in request.

Checksum Generation

According to technology used by merchant, we provide checksum utility and user manual to use this utility.

- Generate Checksum
    - Checksum must include all the parameters i.e. all the mandatory and optional parameters, which are being sent while invoking API.
    - If Merchant code is in Java then
        - Merchant should pass TreeMap of all the parameters (parameter name would be key of TreeMap) to checksum utility method along with key to generate CHECKSUMHASH.

          CheckSumServiceHelper checksumHelper = CheckSumServiceHelper.getCheckSumServiceHelper();

          String checksum = checksumHelper.genrateCheckSum(key, paramMap); // Key : Merchant Key, map : TreeMap of request parameters
    - Else, all the parameters should be passed in alphabetically sorted order on parameter name in the given Checksum utility
    - Checksum utility will be provided separately as per the merchant's platform
- For Server-to-Server calls, pass generated checksum in parameter 'CHECKSUMHASH' and post the parameters on Https call

Verifying Checksum

Paytm sends the generated checksum in the response with parameter name CHECKSUMHASH. The code snippet for verifying checksum will be provided separately.

- Paytm sends the generated checksum in the response with parameter name CHECKSUMHASH

- Once the response is received, the merchant must verify checksum using the utility provided by Paytm.
- If Merchant code is in Java then
    - Merchant should pass TreeMap of all the response parameters (parameter name would be key of TreeMap) to checksum utility method along with key to get boolean response - TRUE/FALSE for matching CHECKSUMHASH

```
CheckSumServiceHelper checksumHelper =
CheckSumServiceHelper.getCheckSumServiceHelper();
boolean isChecksumValid = checksumHelper.verifycheckSum(key, paramMap,
responseCheckSumString);
```

Else, all the parameters should be passed in alphabetically sorted order on parameter name in the given Checksum utility.

Status Query for Transactions

As per secure practices, before marking transaction as success in system on receiving success response from Paytm, merchant should re-verify transaction status and amount of order by calling Paytm status query API from their back-end server.

Paytm status query API returns the ORDERID, TXNAMOUNT, STATUS and some additional parameters for a given order.

If ORDERID and TXNAMOUNT parameters do not match with the order data on merchant side or STATUS is not TXN_SUCCESS, merchant should not fulfill/deliver the order.

The other scenarios where status query will come handy are as follows –

- Paytm returns pending status in response. It happens when bank does not provide the final response to Paytm.
- Due to network drop or Internet issue, merchant does not get the response from Paytm.

For above two scenarios merchant can call Status query API 10 times over a period of 3 days until they get the status as success or failure.

Status Query API for production environment:

https://secure.paytm.in/oltp/HANDLER_INTERNAL/TXNSTATUS

Status Query API for staging environment:

https://pguat.paytm.com/oltp/HANDLER_INTERNAL/TXNSTATUS

Request Parameters: (Should be passed as JSON string in "JsonData" parameter name)

S.No	Parameter Name	Description	Type	Length	Mandatory
1.	MID	This is a unique merchant Id provided to merchant by Paytm at the time of merchant creation.	Alphanumeric		Yes
2.	ORDERID	This is the application transaction Id that was sent by merchant to Paytm at the time of transaction request.	Alphanumeric	50	Yes

https://secure.paytm.in/oltp/HANDLER_INTERNAL/TXNSTATUS?JsonData={"MID":"Paytm MID" , "ORDERID":"Mercahnat order id"}

Response Parameters: (Response will come as JSON string)

S.no	Parameter Name	Description	Type	Mandatory
1.	TXNID	This is a unique Paytm transaction Id that is issued by Paytm for each valid transaction request received from the merchant.	Numeric	Yes
2.	BANKTXNID	The transaction Id sent by the bank (NULL or empty string if the transaction doesn't reaches to the bank).	Alphanumeric	Yes
3.	ORDERID	This is the application transaction Id that was sent by merchant to Paytm at the time of transaction request.	Alphanumeric	Yes
4.	TXNAMOUNT	Amount of transaction.	Numeric	Yes
5.	STATUS	This contains the transaction status and has only two values: **TXN_SUCCESS** **TXN_FAILURE**	Alphanumeric	Yes
6.	TXNTYPE	Any one of below values: **SALE**	Alphanumeric	Yes
7.	GATEWAYNAME	The gateway used by Paytm (ICICI/CITI/WALLET etc).	Alphanumeric	Yes
8.	RESPCODE	This is a numeric transaction response code. All codes refer to a transaction failure or success with each code representing a different reason for failure. Refer to Annexure A for full list.	Alphanumeric	Yes
9.	RESPMSG	This contains a short description of the transaction status. In case of a failed transaction the message will describe the potential reason for the failure.	Alphanumeric	Yes
10.	BANKNAME	Bank name of the card issuing bank.	Alphanumeric	Yes

11.	MID	This is a unique merchant Id provided to merchant by Paytm at the time of merchant creation.	Alphanumeric	Yes
12.	PAYMENTMODE	Mode of Payment. • CC • DC • NB • IMPS • PPI	Alphanumeric	Yes
13.	REFUNDAMT	Total amount refunded till now if merchant has raised any requests.	Numeric	Yes
14.	TXNDATE	Date of transaction.	DateTime	Yes

Response JSON String:

{"TXNID":"62284943","BANKTXNID":"099172","ORDERID":"Test87984","TXNAMOUNT":"1","STATUS":"TXN_SUCCESS","TXNTYPE":"SALE","GATEWAYNAME":"CITI","RESPCODE":"01","RESPMSG":"Txn Successfull.","BANKNAME":"HDFC","MID":"xxxxx34213145601111","PAYMENTMODE":"CC","REFUNDAMT":"1","TXNDATE":"2012-11-09 02:10:29.742447"}