NTT DaTa
Payment Services

Info type: Confidential
Company: NTT Data Payment Services India Limited
Info. owner: Product (PMG)

Transaction Status (Requery) API

CONFIDENTIALITY DISCLAIMER

Document Information

Document Attributes	**Information**
ID	Transaction Status (Requery) API
Owner	Product Management & Engineering
Author	P&E – Product Team

Contents

1. Description

This API is provided to the merchant to track the status of any transaction held via NDPS payment gateway.

Transaction Status Tracking Process:

- Merchant can track the status of the transaction, i.e., initiated/completed (Success/Fail) by end user using the Transaction Status (Requery) API. Merchant sends the MID and transaction details as encrypted data pertaining to transaction details which the status is to be enquired.
- On initiating the Requery API, merchant will receive the status in encrypted response. Merchant will decrypt this response through decryption method.
- Merchant must incorporate the encryption logic provided by NDPS at their end to send the encrypted data in request and to decrypt the encrypted response.

Note*: This API is a **Server-to-Server** Call using **POST** method.

2. Request Format

Transaction Status Sample Request (Open Request-JSON):

- Transaction Status API request **UAT URL**: https://paynetzuat.atomtech.in/ots/payment/status?
- **Production URL** : https://payment1.atomtech.in/ots/payment/status?
- Request and Response of Transaction Status (Requery) API will be encrypted using **AES-512** algorithm.

Request Parameters are to be shared in the format illustrated below:

```
{
 "payInstrument": {
   "merchDetails": {
     "merchId": 1191,
     "merchTxnId": "testQRUPI2",
     "merchTxnDate": "2023-02-28"
 },
   "payDetails": {
     "amount": 1.00,
     "txnCurrency": "INR",
     "signature":
"974d615ad62b1f5b8f08894b0424d6a42356782f12038c8238a928f8f9fb18985b04ac768cf0a7fbbdf34de8d61e58e
9ef061b9860a43fed17eb7fcc2df4b2b8"
   }
 }
```

}

Transaction Status Sample Request (Encrypted Request):

https://payment1.atomtech.in/ots/v2/payment/status?merchId=1191&encData=4CFE8AB36662FB4E664FBFD885E4FA64EB8E7585FDCC2E7117950C0C6A931D726B9BB7D8B7C37504DA04F0801DB17866487A3101AD5419B72FCA8A57BF01F08AADAB7B0953DA41E3538DE3CD883C67044F358C785A36D49339B49628A51D158BFA2E48B9047AA0AEEA8C0C1CD0F38F567465855CC235965FE78D9B6313A3C5F6A275C0774165BEFC9AC79FAD2C5DE35347A2C8C405BC70E4C966BC12B6D421AFABF0D0F080F021A845088EC9491369BEE5B76480A666E95A2182DEEC043168046B46AE390AFC9E5D418540CCCEEFECD81679C733AAD0545A26D11CB657590C8C69C5C42AF0FC6C9E4579A9C6B543572FDAA0AC6E8DA333BF6BC3C4A8C035BB7AF1DCD5AAD494BFF595651428A0341B75827130873139C776C1C37A402EFCB22A0A3BCC7140DA3A48BA3601B05B9779CCF87EE7D9531AB7B8EFF99BDF744A05EAB23B9E3FB04551C9EBAC08878C6B7706187EE257869A0DFDB8666CBEB6B3C043

Sample Encrypted Request Data:

4CFE8AB36662FB4E664FBFD885E4FA64EB8E7585FDCC2E7117950C0C6A931D726B9BB7D8B7C37504DA04F0801DB17866487A3
101AD5419B72FCA8A57BF01F08AADAB7B0953DA41E3538DE3CD883C67044F358C785A36D49339B49628A51D158BFA2E48B90
47AA0AEEA8C0C1CD0F38F567465855CC235965FE78D9B6313A3C5F6A275C0774165BEFC9AC79FAD2C5DE35347A2C8C405BC70
E4C966BC12B6D421AFABF0D0F080F021A845088EC9491369BEE5B76480A666E95A2182DEEC043168046B46AE390AFC9E5D418
540CCCEEFECD81679C733AAD0545A26D11CB657590C8C69C5C42AF0FC6C9E4579A9C6B543572FDAA0AC6E8DA333BF6BC3C4A
8C035BB7AF1DCD5AAD494BFF595651428A0341B75827130873139C776C1C37A402EFCB22A0A3BCC7140DA3A48BA3601B05B9
779CCF87EE7D9531AB7B8EFF99BDF744A05EAB23B9E3FB04551C9EBAC08878C6B7706187EE257869A0DFDB8666CBEB6B3C043

Specifications of the parameters of API Request:

Parent JSON field Name	Parent JSON field Name	Parameter Name	Data Type & Max Length	Sample Input	Dynamic / Static	Content/ Remarks	Mandatory / Optional
payInstrument	merchDetails	merchID	number (10)	1191	Static	Unique ID assigned by NDPS to Merchant	M
		merchTxnId	string (50)	"testQRUPI2"	Dynamic	Unique transaction ID provided by Merchant	M
		merchTxnDate	date (10)	"2023-02-28"	Dynamic	Transaction Date in format yyyy-MM-dd	M
	payDetails	amount	number (12,2)	1.00	Dynamic	Transaction Amount (10 digits prior to the decimal)	M

		txnCurrency	string (5)	"INR"	Static	The currency code of the payment	M
		signature	string (256)	"jhasgdhnsagdagj"	Dynamic	Signature generation using logic provided by NDPS. Encrypt the following sequence - [merchID + merchTxnID + amount + txnCurrency] e.g. : 1191testQRUPI21.06INR	M

3. Response Format:

Response to the transaction status request will comprise of the below illustrated encrypted data.

The response needs to be decrypted as per AES decryption logic provided by NDPS.

Sample Encrypted Response Data:

```
merchId=1191&encData=150C8A5D619A81271AB202298016E54F9867E826683DB7425C9FB43E759B2562900288
15BB8D43135B8B623E0B05F3A48D6C769C4663F573D44F2AF5A9A0ECA2AAEB1EE1F3057FCE8B012BD39567C2E7
3C3ECAAA3CCDF77BC2639239142F214B1D13286E73765071DE6B95E661341D58DCFE60072771D38CB20AC8F4F1
957F70BD2E68531F5978711A097E595094755B173CE0F6C776C329E1D0907AC020ADDB3467DA7BDE5A5766C2C9
E1AE8199F3348492402F7BB6B57E6BE25CCC0F9B181D74C5A8EAB2B2139CEE511C10F69F499F9CD56DAAF69A19
44B343B4C058D16E330F178140B848C4C814E2DF67AAD2C734F3AF08EB0BE4CD5A8C16BB1AC6DAFAF2258EDFA
B210F83B092824DB73EB2868B12149FA7B88E1AAF612090D04CA4A4642CF00944002A2724520132AF956ABAAD6
D22A64028C4C6DB61F1FD224C265F4ECF62C598CEB2090D6A8EADEA2497BE0565390FFD3DE56E0DA99B0D62DF
3E0D96A50EC309BDD788C013DE692EBB3B3C2D9DBEF2879D6F7591DC457FB9727ADFB19069765917E130616E2
6D27DC868BB0F2569E4E80D38B4D1EA19FAC807747CD29533CEAD448C0A6961BE9681CFE954F6F98D0D764B0A
35FD8EC2E7C2CED7CF679B7233DAB57311A9DA49F1C1445D9631E0380F146379093610F37D976811D0F9CF4085
864977E191EB2460CE7233C0C681E3AC957239EB4FED993986B921722F58F879C6ACF9D8A7FFF03981235CB0637
6B9F068DAD680C2F5220DB108DF4407
```

The above response consists of i) merchId, ii) encData

The final response is achieved by decrypting the content of ‘**encData**’ in the above response.

Decryption of Response:

Merchant must pass the encrypted response along with Merchant Specific Response Encryption Key [Pg. 10] and MID in the decryption method as illustrated below:

decryptor = new AtomAES().decrypt(encryptedResponse, Key, iv);

Parameter Name	DataType	Sample Value	Description
decstr	String	BFC23F835C2840C82CCA60671	Encrypted response to the encrypted request triggered, that needs to be decrypted
Key	String	Key provided by NDPS, to decrypt the response	Key provided by NDPS, to decrypt theresponse
IV	String	Same as 'ey'	Same as Key string
dec	String	new.ATOMAES().dec rypt(decstr,key,IV);	Value of this string is an object. That is used to invoke the encrypt function of ATOMAES class. Post encryption, this variable will be appended in the request along with URL, and login.

Note *- The maximum length of the 'key' and 'IV' is '60' and is in 'string' format.

Sample Decrypted Response – Open Data:

Post decrypting the response successfully, merchant will get corresponding data in the below JSON format.

Response Parameters are obtained in the format illustrated below:

```
{
 "payInstrument": {
  "settlementDetails": {
   "reconStatus": "RS"
  },
  "merchDetails": {
   "merchId": 1191,
   "merchTxnId": "testQRUPI2",
   "merchTxnDate": "2023-02-28 23:39:21",
   "clientCode": "007"
  },
  "payDetails": {
   "atomTxnId": 11000155193233,
   "product": "NCA",
   "amount": 1.00,
   "surchargeAmount": 0.06,
   "totalAmount": 1.06
  },
  "payModeSpecificData": {
   "subChannel": "UP",
   "bankDetails": {
    "bankTxnId": "305971152189",
    "otsBankName": "Hdfc Bank",
    "bankAuthId": "65364858680"
   }
  },
  "responseDetails": {
   "statusCode": "OTS0000",
   "message": "SUCCESS",
   "description": "TRANSACTION IS SUCCESSFULL"
  }
 }
}
```

Specifications of API Response:

Parent JSON Field Name	Parent JSON Field Name	Parent JSON Field Name	Parameter Name	Data Type	Sample Input	Max Length	Content/ Remarks
payInstrument	settlementDetails		reconStatus	string	"RS"	10	Reconciliation Status (Refer Possible Values)
	merchDetails		merchID	number	1191	15	Unique ID assigned by NDPS to Merchant
			merchTxnId	string	"testQRUPI2"	50	Unique transaction ID provided by Merchant
			merchTxnDate	datetime	2023-02-28 23:39:21	19	Transaction date must be in yyyy-MM-dd HH:mm:ss format
	payDetails		atomTxnId	number	11000155193233	25	Unique transaction ID (NDPS)

			product	string	"NCA"	50	Product Id provided by NDPS. Passed during the transaction initiation.
			amount	number	1.00	12,2	Amount paid (10 digits prior to decimal)
			surchargeAmount	number	0.06	12,2	Surcharge amount (10 digits prior to decimal)
		prodDetails []	prodName	string	"NCA"	50	Name of the Product
			prodAmount	number	0.00	12,2	Product Specific Amount(10 digits prior to decimal)
			totalAmount	number	1.06	12,2	Total amount [amount + surcharge amount] (10 digits prior to decimal)
	payModeSpecificData		subchannel []	string	"UP"	40	Product used for the Transaction
	bankDetails		bankTxnId	string	"305971152189"	30	Bank Transaction ID
			otsBankName	string	"HDFC Bank"	60	Bank Name
			bankAuthId	string	"65364858680"	45	Bank Auth ID
	responseDetails		statusCode	string	"OTS0000"	30	Status Code
			message	string	"SUCCESS"	30	Status Message
			description	string	"TRANSACTION IS SUCCESSFUL"	50	Status Description

Possible Values for parameter 'reconStatus':

1. RS – Reconciled Settled (Reconciled and settled to merchant)
2. RNS – Reconciled Not Settled (Reconciled but not settled to merchant)
3. NRNS – Not Reconciled Not Settled (Prior to reconciliation)
4. PNRNS – Not Reconciled Not Settled (Not reconciled and not settled to merchant on T0 settlement)
5. PNRS – Payment Not Reconciled Settled (Not reconciled but settled to merchant)

Response Codes:

Error Code	Message	Description
OTS0000	SUCCESS	TRANSACTION IS SUCCESSFUL / FORCE SUCCESS *
OTS0002	FORCE SUCCESS	TRANSACTION IS FORCE SUCCESS
OTS0101	CANCEL	TRANSACTION IS CANCELLED BY USER ON PAYMENT PAGE
OTS0201	TIMEOUT	TRANSACTION IS TIMEOUT
OTS0401	NODATA	NO DATA
OTS0451	INVALIDDATA	INVALID DATA
OTS0501	INVALIDDATA	INVALID DATA
OTS0600	FAILED	TRANSACTION IS FAILED / AUTO REVERSAL *
OTS0301	INITIALIZED	TRANSACTION IS INITIALIZED
OTS0351	INITIATED	TRANSACTION IS INITIATED
OTS0551	PENDING	TRANSACTION IS PENDING
OTS0951	SOMETHING WENT WRONG	UNEXPECTED ERROR

* Either of the mentioned description will be received.

Points to be Noted*-

- 'OTS401' i.e., 'NO DATA' means that the transaction data is not available due to incorrect input.
- Requery API will fetch the status only within **30** days from the day of transaction.
- The description will be 'FORCE SUCCESS' if force success is enabled & as 'AUTO REVERSAL' if auto reversal is enabled. Same can be configured by contacting NDPS backend team. (For scenarios, refer to Callback API document).
- Merchant can either opt for "**Auto Reversal**" or "**Force Success**", below is the pre-condition and the implication of each respectively –
  i) Auto Reversal : Amount is debited, but transaction is terminated, then the status is implicitly converted to Auto Reversal, implying the transaction was failed (amount gets reversed to the payer bank account).
  ii) Force Success : Amount is debited, but transaction is terminated, then the status is implicitly converted to Force Success, implying the transaction as successful (amount gets settled to merchant bank account).
- In 'Force Success' scenario, the response in 'Callback' will be received as 'Success' only.

4. AES Encryption Logic:

- Transaction Status (Requery) API's request and returned response are shared post AES-512 encryption.
- The following KEYs are to be used for UAT:

MerchId	encResKey	encReqKey
9135	58BE879B7DD635698764745511C704AB	7813E3E5E93548B096675AC27FE2C850

AES Encryption Java Code:

```
import java.util.logging.Logger;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

public class AtomEncryption {
static Logger log = Logger.getLogger(AtomEncryption.class.getName());

private static int pswdIterations = 65536;
private static int keySize = 512;
private static final byte[] ivBytes = {
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
};

public static String encrypt(String plainText, String key) {
        try {
                byte[] saltBytes = key.getBytes("UTF-8");
                SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
                PBEKeySpec spec = new PBEKeySpec(key.toCharArray(), saltBytes, pswdIterations,
keySize);
                SecretKey secretKey = factory.generateSecret(spec);
                SecretKeySpec secret = new SecretKeySpec(secretKey.getEncoded(), "AES");

                IvParameterSpec localIvParameterSpec = new IvParameterSpec(ivBytes);
                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(1, secret, localIvParameterSpec);
                byte[] encryptedTextBytes = cipher.doFinal(plainText.getBytes("UTF-8"));
                return byteToHex(encryptedTextBytes);
        } catch (Exception e) {
                log.info("Exception while encrypting data:" + e.toString());
        }
        return null;
}

public static String decrypt(String encryptedText, String key) {
        try {
                byte[] saltBytes = key.getBytes("UTF-8");
                byte[] encryptedTextBytes = hex2ByteArray(encryptedText);
```

```
			SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
			PBEKeySpec spec = new PBEKeySpec(key.toCharArray(), saltBytes, pswdIterations,
keySize);
			SecretKey secretKey = factory.generateSecret(spec);
			SecretKeySpec secret = new SecretKeySpec(secretKey.getEncoded(), "AES");
			IvParameterSpec localIvParameterSpec = new IvParameterSpec(ivBytes);
			Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
			cipher.init(2, secret, localIvParameterSpec);
			byte[] decryptedTextBytes = (byte[]) null;
			decryptedTextBytes = cipher.doFinal(encryptedTextBytes);
			return new String(decryptedTextBytes);
		} catch (Exception e) {
			log.info("Exception while decrypting data:" + e.toString());
		}
		return null;
}

private static String byteToHex(byte[] byData) {
		StringBuffer sb = new StringBuffer(byData.length * 2);
		for (int i = 0; i < byData.length; ++i) {
			int v = byData[i] & 0xFF;
			if (v < 16)
				sb.append('0');
			sb.append(Integer.toHexString(v));
		}
		return sb.toString().toUpperCase();
}

private static byte[] hex2ByteArray(String sHexData) {
		byte[] rawData = new byte[sHexData.length() / 2];
		for (int i = 0; i < rawData.length; ++i) {
			int index = i * 2;
			int v = Integer.parseInt(sHexData.substring(index, index + 2), 16);
			rawData[i] = (byte) v;
		}
		return rawData;
}
public static void main(String[] args) {

		try {
			String encryptedData = AtomEncryption.encrypt("1235", "ASWKLSLLFS4sd4g4gsdg");
			System.out.println("encryptedData : " + encryptedData);
		} catch (Exception e) {
			// TODO: handle exception
		}
}
}
```

Signature Generation Logic:

i) For any given transaction, the signature for Transaction Status API's request and the response is to be generated using the shared hashing code below.

- Signature generation sequence [merchID + merchTxnID + amount + txnCurrency]
  e.g. : 1191testQRUPI21.06INR
- The UAT request and response hash keys are as follows :

MerchId	reqHashKey	respHashKey
9135	ea59e6ee036c81d8b5	ea59e6ee036c81d8b6

Signature Generation (Hashing) Java Code:

```
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class AtomSignature {
public static String generateSignature(String hashKey, String[] param) {
        String resp = null;
        StringBuilder sb = new StringBuilder();
        for (String s: param) {
                sb.append(s);
        }

        try {
                System.out.println("String =" + sb.toString());
                resp = byteToHexString(encodeWithHMACSHA2(sb.toString(), hashKey));
        } catch (Exception e) {
                System.out.println("Unable to encocd value with key :" + hashKey + " and input :" +
sb.toString());
                e.printStackTrace();
        }
        return resp;
}
private static byte[] encodeWithHMACSHA2(String text, String keyString)
throws NoSuchAlgorithmException, InvalidKeyException, UnsupportedEncodingException {
        Key sk = new SecretKeySpec(keyString.getBytes("UTF-8"), "HMACSHA512");
        Mac mac = Mac.getInstance(sk.getAlgorithm());
        mac.init(sk);
        byte[] hmac = mac.doFinal(text.getBytes("UTF-8"));
        return hmac;
}
public static String byteToHexString(byte byData[]) {
        StringBuilder sb = new StringBuilder(byData.length * 2);
        for (int i = 0; i < byData.length; i++) {
                int v = byData[i] & 0xff;
                if (v < 16)
```

```
                        sb.append('0');
                sb.append(Integer.toHexString(v));
        }
        return sb.toString();
}
```

UAT environment details:

The UAT environment details are as follows :

UAT IP : 13.127.25.237

UAT Server : NDPS IP to be whitelisted at the merchant’s end (if needed) so that response can be posted.