

Specifications of API Response	11
Response Codes.....	12
AES Encryption Logic	13
AES Encryption Java Code.....	14
Signature Generation Logic	16
Signature Generation (Hashing) Java Code	17
UAT Environment Details	18

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.

Request Format

Transaction Status Sample Request (Open Request-JSON)

- Transaction Status API request
- **UAT URL:** paynetzuat.atomtech.in/ots/payment/status
- **Production URL :** 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": {
    "headDetails": {
      "api": "TXNVERIFICATION",
      "source": "OTS"
    },
    "merchDetails": {
      "merchId": 446442,
      "password": "Test@123",
      "merchTxnId": "67a44ce2ed4c6",
      "merchTxnDate": "2025-02-06"
    },
    "payDetails": {
      "atomTxnId": "11000000631738",
      "amount": 10.00,
      "txnCurrency": "INR",
      "signature":
        "f6baf5a945f3f1f0db9d02e04df7fd5a287a7f35b50375f700894196c340b4a82e89c13e1e0078ba1b974f67d917c1c609
        46888002a3be8c867e715f7da6c68c"
    }
  }
}
```

Transaction Status Sample Request (Encrypted Request)

<https://paynetzuat.atomtech.in/ots/payment/status?merchId=446442&encData=FD35EABE4985D0E49C71B51BDF35C809F4187B9088630B4C4128E281F1283861AF7AF4AE052E633B1738FD2E233184A8C93C0AD A66D00EC582BDCE7C175ABED9C53B8BFEF92AD548A9E771874368152A757A99E122029C4B6A63DE4DB7E09869102B49A7B47312EF6ADE413686F71D589ED83B7466B6A69DB5317C244C20C4E5DA9E02B55F23692B9F11673116CE04ADEF4EA10219D4345E3D0D8BADBD01299EAEBO6E8696AA6CA9B60BF3C5DFA009811C477CF90E7BEF53A0BCA37429CBFF2F4FC9C97FFC1F3DFE571E8E1058CA11344A5C1EA2FBDAA773119EFD23AADAABA2E475A247398F293C4512A2AFF9BC0B8154BF107850DB89688CE6CCBF8F40947D3CEE3ABA0431BAABB897748409FA0D68D4D91FCB8C2DB9D88B3FACAD907D1E174EBF8641EB544C28780E0B2BC4EBC2B79CD6E5B114B0B1A10CC38F12259623E4650784ECE6576FBC8EDC6CAEF78C607E619756CC9DE359AA1E6CD1F7D8A5A694AB695339603ED4EE86196B2832E6080F5496292E866E02DED3A6186BF6FB839>

37B4424AC101E40CEDBD53B871E677333B145C7B33542D7B01D0473E51A408249A7D56A341C4BCB0300D56EF9FFB44E0FE4D621945656CB8C7835BE1C9F5E283262F9C0A90C6EA70895BD7C36A526FD9

Sample Encrypted Request Data

FD35EABE4985D0E49C71B51BDF35C809F4187B9088630B4C4128E281F1283861AF7AF4AE052E633B1738FD2E233184A8C93C0ADA66D00EC582BDCE7C175ABED9C53B8BFEF92AD548A9E771874368152A757A99E122029C4B6A63DE4DB7E09869102B49A7B47312EF6ADE413686F71D589ED83B7466B6A69DB5317C244C20C4E5DA9E02B55F23692B9F11673116CE04ADEF4EA10219D4345E3D0D8BADBD01299EAEBO6E8696AA6CA9B60BF3C5DFA009811C477CF90E7BEF53A0BCA37429CBFF2F4FC9C97FFC1F3DFE571E8E1058CA11344A5C1EA2FBDAA773119EFD23AADAABA2E475A247398F293C4512A2AFF9BC0B8154BF107850DB89688CE6CCBF8F40947D3CEE3ABA0431BAABB897748409FA0D68D4D91FCB8C2DB9D88B3FACAD907D1E174EBF8641EB544C28780E0B2BC4EBC2B79CD6E5B114B0B1A10CC38F12259623E4650784ECE6576FBC8EDC6CAEF78C607E619756CC9DE359AA1E6CD1F7D8A5A694AB695339603ED4EE86196B2832E6080F5496292E866E02DED3A6186BF6FB83937B4424AC101E40CEDBD53B871E677333B145C7B33542D7B01D0473E51A408249A7D56A341C4BCB0300D56EF9FFB44E0FE4D621945656CB8C7835BE1C9F5E283262F9C0A90C6EA70895BD7C36A526FD9

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	headDetails	api	String (20)	TXNVERIFICATION	Static	For Transaction verification "TXNVERIFICATION" fixed	M
		source	String (3)	OTS	It has to be only "OTS"	It's static, only OTS	M
	merchDetails	merchID	number (10)	446442	Static	Unique ID assigned by NDPS to Merchant	M

		merchTxnId	string (50)	"67a44ce2ed4c6"	Dynamic	Unique transaction ID provided by Merchant	M
		merchTxnDate	date (10)	"2025-02-06"	Dynamic	Transaction Date in format yyyy-MM-dd	M
	payDetails	atomTxnId	number (25)	11000000631738	Dynamic	NDPS Transaction ID	M
		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)	"f6baf5a945f3f1f0db9d02e04df7fd5a287a7f35b50375f700894196c340b4a82e89c13e1e0078ba1b974f67d917c1c60946888002a3be8c867e715f7da6c68c"	Dynamic	Signature generation using logic provided by NDPS. Encrypt the following sequence - [merchID + Password + merchTxnID + amount + txnCurrency + api] e.g. : 446442Test@12367a44ce2ed4c610.00INRTXNVERIFICATION	M

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

```
encData=0CD63904F407DC4D24167F8D8F2BEE996E7F85C37096313309FD9A634B662770E3A62265D92E
6B1E941D1CE3E17FFCBBB5823977AAFFE6CEE0B7DC92F1CABDC3EC99A6A3A0B4F1AA86A30F0633FDA75
A3D2881321BA56166F1FDB3BF9AC8D170F8623183F6BD241B4ACF2344921670EB05799DA2B883D5AD2D
6F15EE62F4D424BBB5EA812A74C7B572ABB7257C9A88936BEC29855C4520F99F9CE4A8B5BA89B86CDF4
211C9B5BF6F7960339F15B7A8DE17A92E16C0CCD7E7D071A174B38B2AA0036623E35B55E13F5775F5459
6F84C5DC25640EA48E260387FC0EC0C871E11AA512916495C938D3ED765FCA2C2B4EFAF6E9BFCCD6C777
6F6BE997EA6CF9F28A3DF14393B1A10A2C4CEE6491EFFF58B3CB27E83F3D74FB614E928E0E1D3555FDB5
69253BFB9F3AFAE522F7BB15096BC440B4B802D697C99AB264C9E66FEDCF1F986350ADA6020D17B84579
B6335235D6182AA7C3A4AFC011988ABEFC9B454C85F5861F0FAD79F0E1E1BEDB7A89B988CBE4210FF159
E4D5496BB37FFFB364B0B5AD00FC9425E72CC02861D5F169C313C1F804AA98FCA621264CD4900AF63D0
5A26289A99F136AD55E45893B38A8AC4C3240F32A2F072E7157ED0FAEF010C3C438400879901958BF446
479C74A303DE274EFF9CC32D43DFC20EA0D727A6496A0673EAC02E27D66431787371078D28752249060
16253DDCC7ADD197A321A49B5B83C&merchId=446442
```

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:

```
String plainText =AtomEncryption.decrypt(encryptedData, key);
```

Parameter Name	Data Type	Sample Value	Description
encryptedData	String	BFC23F835C2840C8 2CCA60671	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 the response
decrypt	String	new. AtomEncryption ().dec ryp t(decstr,key);	Value of this string is an object. That is used to invoke the decrypt function of AtomEncryption class.

Note *- The maximum length of the 'key' 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": "NRNS"
      },
      "merchDetails": {
        "merchId": 446442,
        "merchTxnId": "67a44ce2ed4c6",
        "merchTxnDate": "2025-02-06 11:17:50"
      },
      "payDetails": {
        "atomTxnId": 11000000631738,
        "product": "NSE",
        "amount": 10.00,
        "surchargeAmount": 0.00,
        "totalAmount": 10.00
      }
    }
  ]
}
```

```

},
"payModeSpecificData": {
  "subChannel": "NB",
  "bankDetails": {
    "bankTxnId": "NBnL0kjrRRpBlr2KYVR5",
    "otsBankName": "Atom Bank"
  }
},
"responseDetails": {
  "statusCode": "OTS0000",
  "message": "SUCCESS",
  "description": "TRANSACTION IS SUCCESSFUL"
}
}
]
}

```

Specifications of API Response

Parent JSON Field Name	Parent JSON Field Name	Parent-JSON Field Name JSON Field Name	Parameter Name	Data Type	Sample Input	Max Length	Content/ Remarks
payInstrument	settlementDetails		reconStatus	string	"NRNS"	10	Reconciliation Status (Refer Possible Values)
	merchDetails		merchID	number	446442	15	Unique ID assigned by NDPS to Merchant
			merchTxnId	string	"67a44ce2ed4c6"	50	Unique transaction ID provided by Merchant
			merchTxnDate	datetime	2025-02-06 11:17:50	19	Transaction date must be in yyyy-MM-dd HH:mm:ss format
	payDetails		atomTxnId	number	11000000631738	25	Unique transaction ID (NDPS)
			product	string	"NSE"	50	Product Id provided by NDPS. Passed during the transaction initiation.
			amount	number	10.00	12,2	Amount paid (10 digits prior to decimal)
			surchargeAmount	number	0.00	12,2	Surcharge amount (10 digits prior to decimal)
		prodDetail(s)	prodName	string	"NCA"	50	Name of the Product

			prodAmount	number	10.00	12,2	Product Specific Amount(10 digits prior to decimal)
			totalAmount	number	10.00	12,2	Total amount [amount + surcharge amount] (10 digits prior to decimal)
	payModeSpecificData		subchannel []	string	"NB"	40	Product used for the Transaction
	bankDetails		bankTxnId	string	"NBnL0kjRRpBlr2KYVR5"	30	Bank Transaction ID
			otsBankName	string	"Atom 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	For Merchant's/End User's Understanding
OTS0000	SUCCESS	CHALLAN GENERATED SUCCESSFULLY / TRANSACTION IS SUCCESSFUL	SUCCESS
OTS0002	SUCCESS	TRANSACTION IS FORCE SUCCESS	SUCCESS
OTS0201	FAILED	TRANSACTION IS TIMEOUT	PENDING
OTS0301	INITIALIZED	TRANSACTION IS INITIALIZED	PENDING
OTS0351	INITIATED	TRANSACTION IS INITIATED	PENDING

OTS0551	PENDING FROM BANK	TRANSACTION IS PENDING FROM BANK	PENDING
OTS0001	FAILED	TRANSACTION IS AUTO REVERSAL	FAILED
OTS0101	CANCELLED	TRANSACTION IS CANCELLED BY USER ON PAYMENT PAGE	FAILED
OTS0401	NODATA	Data not found	FAILED
OTS0503	NODATA	API name should not exceed 20 characters	FAILED
OTS0504	FAILED	Please provide valid apiname	FAILED
OTS0506	FAILED	Signature Mismatched	FAILED
OTS0507	FAILED	[merchTxnDate should be in yyyy-MM-dd format]	FAILED
OTS0508	FAILED	Invalid Transaction Currency	FAILED
OTS0509	FAILED	Invalid Transaction Amount	FAILED
OTS0510	FAILED	Invalid Transaction Date	FAILED
OTS0522	FAILED	Invalid Password	FAILED
OTS0600	ABORTED/FAILED	TRANSACTION IS ABORTED / TRANSACTION IS FAILED	FAILED
OTS0951	FAILED	SOMETHING WENT WRONG	FAILED

* 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 –
 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).
 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.

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	encReqKey	encResKey
446442	A4476C2062FFA58980DC8F79EB6A799E	75AEF0FA1B94B3C10D4F5B268F757F11

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 = 256;
    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 locallyIvParameterSpec = new IvParameterSpec(ivBytes);
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

```

cipher.init(1, secret, locallyParameterSpec);
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 locallyParameterSpec = new IvParameterSpec(ivBytes);
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(2, secret, locallyParameterSpec);
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 + Password + merchTxnID + amount + txnCurrency + api**]
Eg. - 446442Test@12367a44ce2ed4c610.00INRTXNVERIFICATION
- AT request and response hash keys are as follows :

MerchId	reqHashKey	respHashKey
446442	KEY123657234	KEYRESP123657234

- Encrypted –

f6baf5a945f3f1f0db9d02e04df7fd5a287a7f35b50375f700894196c340b4a82e89c13e1e0078ba1b974f67
d917c1c60946888002a3be8c867e715f7da6c68c

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[] hmacmac.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.