**Transactional Data Service API Documentation**

**Overview**

The Transactional Data Service API provides a secure and efficient way for clients to send transactional data for processing. This service uses JSON for data interchange and requires proper validation for each request.

**API URL Structure**

https://api.transactiondataservice.com/api/v/1/{apiKey}?m={method}

* **Base URL**: **https://api.transactiondataservice.com/**
* **Version**: **/v1/**
* **Endpoint**: **/transactions/**
* **API Key**: **{apiKey}** - A unique identifier provided to each client for authentication
* **Parameter “m”**: **{method}** – The identifier for the current transactional method, in this case it should be “netcashach”

**Example**: https://api.transactiondataservice.com/api/v/1/A0RlX5HMCwfSQ2y70ZsCnTDEx8zY5QhJ?m=netcashach

**Request Format**

**Specifications**

* **HTTP Request**: **POST**
* **Query Parameters**: **method**
* **Request body**: **JSON formatted**
* **Request body content-type**: **application/json**

**JSON Document Structure**

Each request must be a JSON document containing two main components: validation data and transactional data.

****

**Validation Data**

* **checksum**: A string generated by concatenating several specified fields in the request with a secret word. This is used to verify the integrity and authenticity of the request. The checksum is the result of an SHA1 encoding applied to the following concatenation: *customeraccount+amount+currency+routing+email+trxtype+merchant+{secret word}*
  + **example**: if the secret word provided is “nvcobuvrlt”, the complete string for the sample data above would be: “X750295999.99usd000023923emailid@somedomain.compayoutlatcorpnvcobuvrlt”  **SHA1 checksum: 160681555b695aaf8a936b2521fde95dff440f20**
* **email**: The email address for the user associated with the merchant account.

**Transactional Data**

* **id** (required): A unique identifier for the transaction on your system. Only one transaction is allowed with any given value in this field. Type: string.
* **customeraccount** (required): The customer’s unique identifier, usually the username or id
* **amount** (required): The monetary value of the transaction. Should contain only digits and “dot” as the decimal separator. Type: string.
* **currency** (required): The currency in which the transaction is made, according to the ISO 4217 three-letter standard. Type: string.
* **trxtype** (required): Type of the transaction, possible values: “payout”. Type: string.
* **cxname** (required): Name of the customer. Type: string.
* **routing**: Routing number for the bank account. Type: string.
* **bankaccount** (required): Bank account number. Type: string.
* **accounttype** (required): Type of bank account, example “c” for “checking”. Type: string.
* **email** (required): Email of the customer. Type: string.
* **address** (required): Address of the customer according to the bank records. Type: string.
* **parent** (required): Represents the channel of the transaction, it must be “direct”. Type: string.
* **merchant** (required): Name identifier of the merchant. Type: string.
* **comment** (required): Any comment you may wish to include. Type: string.

**Response Format**

**Successful Request**

****

* **status**: Indicates the success of the request.
* **message**: Descriptive message about the request status.
* **reference**: A unique number assigned to the transaction for tracking purposes.

**Failed Request Examples**

****

* **status**: Indicates the failure of the request.
* **message**: Generic error message
* **error**: Specific error description
* **ext**: In some cases, additional details about the error may be provided.

**Error Handling**

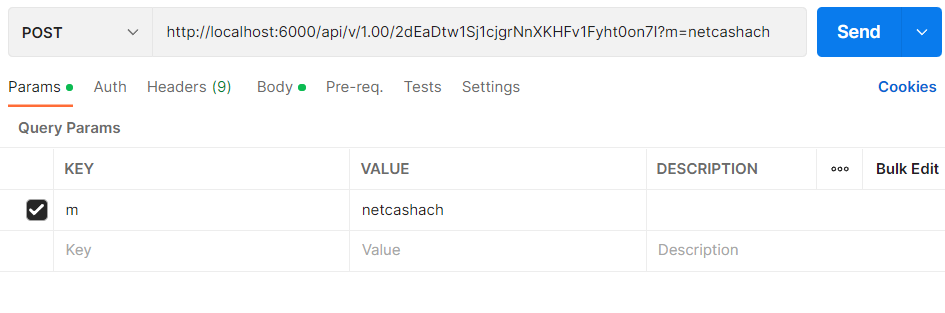
Clients should handle common HTTP status codes such as 400 (Bad Request), 401 (Unauthorized), 500 (Internal Server Error), etc. Each error response will contain a descriptive message and an error code for more details.

**Integration Tips**

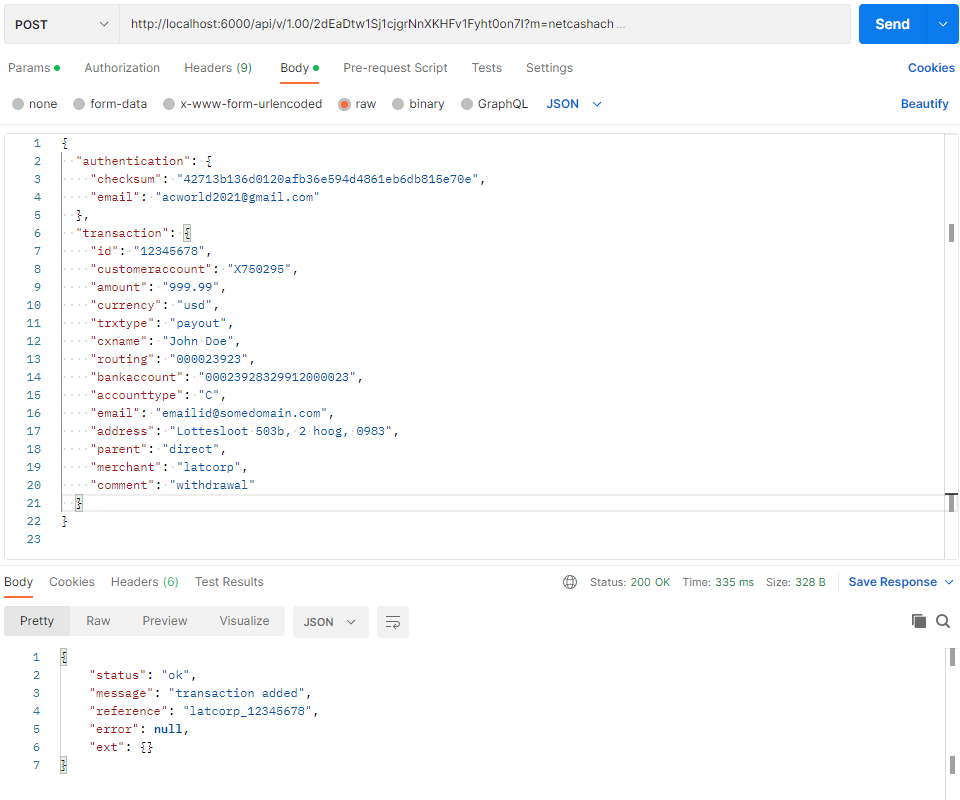
* Ensure that the checksum is correctly generated for each request.
* Validate the required fields before sending the request.
* Monitor response status and error codes for effective error handling and debugging.
* Sanitize your data before submitting it to our system, otherwise some strings may be identified as containing malicious code.

**Postman Example**

*Post parameters section:*



*Body section:*



**Security Recommendations**

* Protect your API key and secret word. Do not expose them in client-side code.
* Use HTTPS to ensure the security of data in transit.
* Regularly rotate your API key and secret word.

**Support**

For integration support or any queries, contact our support team at aiepicstudio@engineer.com.