Service & API Documentation

v1.06

Questions?

developers@tsgglobal.com

Document History

Revision	Date	Description	Author
1.00	June 21, 2021	Initial document.	ACE
1.01	August 4, 2021	Updated API endpoints	ACE
1.02	January 18, 2022	Added Postman and Quick Start	ACE
1.03	January 25, 2022	Added curl examples	ACE
1.04	January 26, 2022	Added .NET examples	ACE
1.05	January 27, 2022	Formatting and layout	ACE
1.06	February 3, 2022	Added Create examples	ACE

Definitions

Below is a list of definitions to help you better understand the contents of this document:

Brand

The company or entity the consumer believes to be sending the message.

Example: the brand would be “Dunder Mifflin” for any messages that would say “Thanks for subscribing to the Dunder Mifflin paper stock alert system.”)

Campaign

The use-cases associated with a phone number (e.g. two-factor) – currently relating to 10DLC, or 10-digit long code numbers. Campaigns have different surcharges based on use-case, brand score, message class, and different through-puts based on the type of Campaign.

Example: a Campaign for brand “Dunder Mifflin” may be a low-volume mixed campaign, that includes messages relating to two-factor authentication (2FA) and some basic marketing.

Chaincode
Chaincode is programmable code, written in Go, Java, or node.js, and instantiated on a channel within the blockchain network. Developers use chaincode to develop business contracts, asset definitions, and collectively-manage decentralized applications. The chaincode manages the ledger state through transactions invoked by applications. Assets are created and updated by a specific chaincode and cannot be accessed by another chaincode.

Organization
A business that is associated with a phone number in the chain-of-custody. This can be a carrier, reseller, enterprise company, or the brand associated with a number. All businesses within TNID have used phone numbers and have a business rating (score) that reflects their fair use of industry resources.

Partner
Represents your downstream customer – an organization that purchases products from you and leverages phone numbers that you have acquired.

Introduction
This document describes the functional design of the Telephone Number IDentity (TNID) management application and associated GraphQL API end-points. TNID supports real-time (HTTP) data write and lookup services, as well as upcoming HTTP (webhook) access to retrieve notifications as described in this document.

History
TNID was launched in 2021 to provide the communications industry a streamlined phone number data management system, powered by a private, decentralized blockchain network through an open-source framework provided through Hyperledger Fabric. TNID is an aggregation service interconnected with dozens of centralized, independent data repositories that carriers and enterprise businesses use every day. TNID lives "over-the-top" of these services and utilizes these 3rd party external-facing APIs to streamline the number asset management process and reduce development time for businesses looking to use telephone numbers at scale.

The initial deployment of TNID provides an enhanced 10DLC campaign management system that interconnects with The Campaign Registry, allowing organizations to manage both brands and campaigns associated with the 10DLC ecosystem. TNID provides enhanced functionality by:

- providing native cross-organization transparency into brands and campaigns

- transmitting data directly to The Campaign Registry, the NetNumber Override Service Registry (OSR), as well as wireless carriers (e.g. T-Mobile) through Syniverse
- allowing for campaign and brand tagging, as well as the ability to quickly duplicate and modify existing campaigns with additional phone numbers

Services

Today, we allow users to manage data contained within TNID via three different methods:

GraphQL API

HTTP provisioning services described in this document are available via connection to TNID cloud infrastructure centers located in the United States.

- Queries are used to read or fetch data values.
- Mutations are used to write, post, or update values.

We provide a GraphiQL (pronounced "graphical") Explorer to examine our APIs. The GraphiQLExplorer enables you to interactively construct full queries by clicking through available fields and inputs without the repetitive process of typing these queries out by hand. You can find our GraphiQL Explorer and data schemas here: https://app.tnid.com/graphql

Portal / Graphical User Interface (GUI)

A React-based framework front-end is currently available for general use. The acceptance environment portal is available here: https://acc.tnid.com The production environment portal is available here: https://app.tnid.com

Blockchain Node Service

TNID data is also manageable via an immutable, private blockchain network. TNID currently commits some (but not all) created data to a Hyperledger Fabric ledger ("on-chain").

We can invite you to the network via Amazon Web Services' Managed Blockchain solution – we simply require your AWS account ID, as well as the email address of a technical resource. If you do not use AWS, you can still connect to the network either through an on-prem solution or through a different hosted cloud provider.

Read more about Hyperledger Fabric:
https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html

Read more about AWS Managed Blockchain:
https://aws.amazon.com/managed-blockchain/

GraphQL API Overview

TNID supports the provisioning and retrieval of data through a GraphQL API. You can, if you choose, interact with our GraphQL as you would a RESTful API using a POST. You can read more about GraphQL here: https://graphql.org/

General Operations

Below is a list of operations that you can perform via the GraphQL API we provide:

Queries

- organization(s)
  - Query basic information about one or more Organizations (if you have their unique ID).
- brand(s)
  - Query information about one or multiple Brands (provided by customers and data in The Campaign Registry) and the Organization associated with that brand.
- campaign(s)
  - Query information about one or more campaigns Campaign (provided by customers and data in The Campaign Registry), and the Brand associated with that campaign.
- usecases
  - Query a list of use-cases that are provided by The Campaign Registry in relation to 10DLC.
- tnidHistory
  - Query any historical changes to a phone number record/asset and view the chain-of-custody associated with a phone number (where you stand in the chain, and those above and below you are not hidden).
- searchNumbers
  - Query information about a phone number, including any NNID, Brand, or Campaign associated with it.
- event(s)
  - Query recent Events that (may) affect your Organization, such as a new Brand being created, a new Partner being registered, or a new Campaign being generated.
- statistics
  - View basic information relating to your Orgs active campaigns, brands, numbers, and pending Events.

Mutations

- createPartner
  - Create a child Organization underneath your Organization that represents a customer of yours.
    - Up to 10 “levels” in the chain-of-custody are currently supported.
- updateOrganization

    - Update basic business information, and contact information.
- uploadOrganizationLogo
    - Upload an image and associate it with an Organization.
        - Recommended 500px square.
- createBrand
    - Create a Brand using re-using many of the basic fields required to create an Organization. Organizations may have one or more Brands, and there is no limit currently to the number of Brands an Organization may have.
- createBrandNonBlocking
    - [Pending]
- updateBrand
    - Update basic information relating to a Brand.
- updateBrandLogo
    - Update the logo associated to a Brand
        - Recommend 500px square.
- deleteBrand
    - Remove a Brand from TNID
- createCampaign
    - Create a Campaign in the Campaign Registry and associate phone numbers with the Campaign once it is approved.
    - Phone numbers can be supplied as a comma-separated list as a string (e.g. "["14252437709","14252437710"]" in order to be accepted.
- updateCampaign
    - Update some fields relating to a Campaign (that The Campaign Registry allows without having to delete and start over) or add more tags to a Campaign.
- deactivateCampaign
    - Deactivate a Campaign and remove it/end it in The Campaign Registry.

Subscriptions

- organizationAdded
    - Receive a JSON-formatted notification whenever a new organization/partner is added.
- organizationUpdated
    - Receive a JSON-formatted notification whenever an Organization or Partner is updated.
- campaignAdded
    - Receive a JSON-formatted notification whenever a new Campaign is added.
- brandAdded
    - Receive a JSON-formatted notification whenever a new Brand is added.
- brandUpdated
    - Receive a JSON-formatted notification whenever a Brand is updated.
- brandDeleted

- Receive a JSON-formatted notification whenever a Brand is deleted.

GraphQL API Quick Start Guide

The TNID API uses GraphQL as its interface. GraphQL uses only one endpoint and only POST requests are valid. This document contains information to authenticate and to execute queries and mutations using Postman, cURL, .net and other languages/frameworks.

Variables

The following table lists variables that are common between the examples. The variable values must be substituted with the current values that can differ from customer to customer and between the different environments (acceptance and production).

Environment	Variable	Value
Acceptance	[Token_Url]	http://ci-tnid-portal-keycloak-acc-westus-001.westus.azurecontainer.io:8080/auth/realms/tnid/protocol/openid-connect/token
Acceptance	[GraphQL_Endpoint]	https://acc.tnid.com/graphql
Acceptance	Schema endpoint	https://acc.tnid.com/graphql?sdl
Acceptance	[Client_ID]	tnidgraphqlapiclient
Acceptance	[Client_Secret]	<Your client secret>, received from support
Acceptance	[User_Name]	<Your username>, received from support
Acceptance	[Password]	<Your password>, received from support
Production	[Token_Url]	http://ci-tnid-portal-keycloak-prd-westus-001.westus.azurecontainer.io:8080/auth/realms/tnid/protocol/openid-connect/token
Production	[GraphQL_Endpoint]	https://app.tnid.com/graphql
	Schema endpoint	https://app.tnid.com/graphql?sdl
Production	[Client_ID]	tnidgraphqlapiclient
Production	[Client_Secret]	<Your client secret>, received from support
Production	[User_Name]	<Your username>, received from support
Production	[Password]	<Your password>, received from support

Authorization

Every call to the TNID endpoint must be authenticated and authorized. This is implemented using a *Bearer Token*.

Access token
To request a token, a call must be made to a *token endpoint*, using a number of credentials. When the credentials are validated, the endpoints responds with an *access token* with a limited lifetime (300 seconds or 5 minutes). After that a new *access token* must be requested.

Refresh token
To prevent supplying a username and password every time the token endpoint is called, the token endpoint can also supply a *refresh token* that can be supplied instead of the user/password credential. To indicate that the *refresh token* is used, the parameter *grant_type* must have the value refresh_token while in the case of user/password credentials the value password is used.

Common Errors
Below are some common errors that can be returned, and how to fix them:

Token endpoint

Response:
```
{"error":"invalid_grant","error_description":"Invalid user credentials"}
```

Resolution:
Check your username and/or password.

Response:
```
{"error":"unauthorized_client","error_description":"Invalid client secret"}
```

Resolution:
Check if the correct client secret is used. These differ between the Acceptance and the Production environment.

Response:
```
{"error":"invalid_client","error_description":"Invalid client credentials"}
```

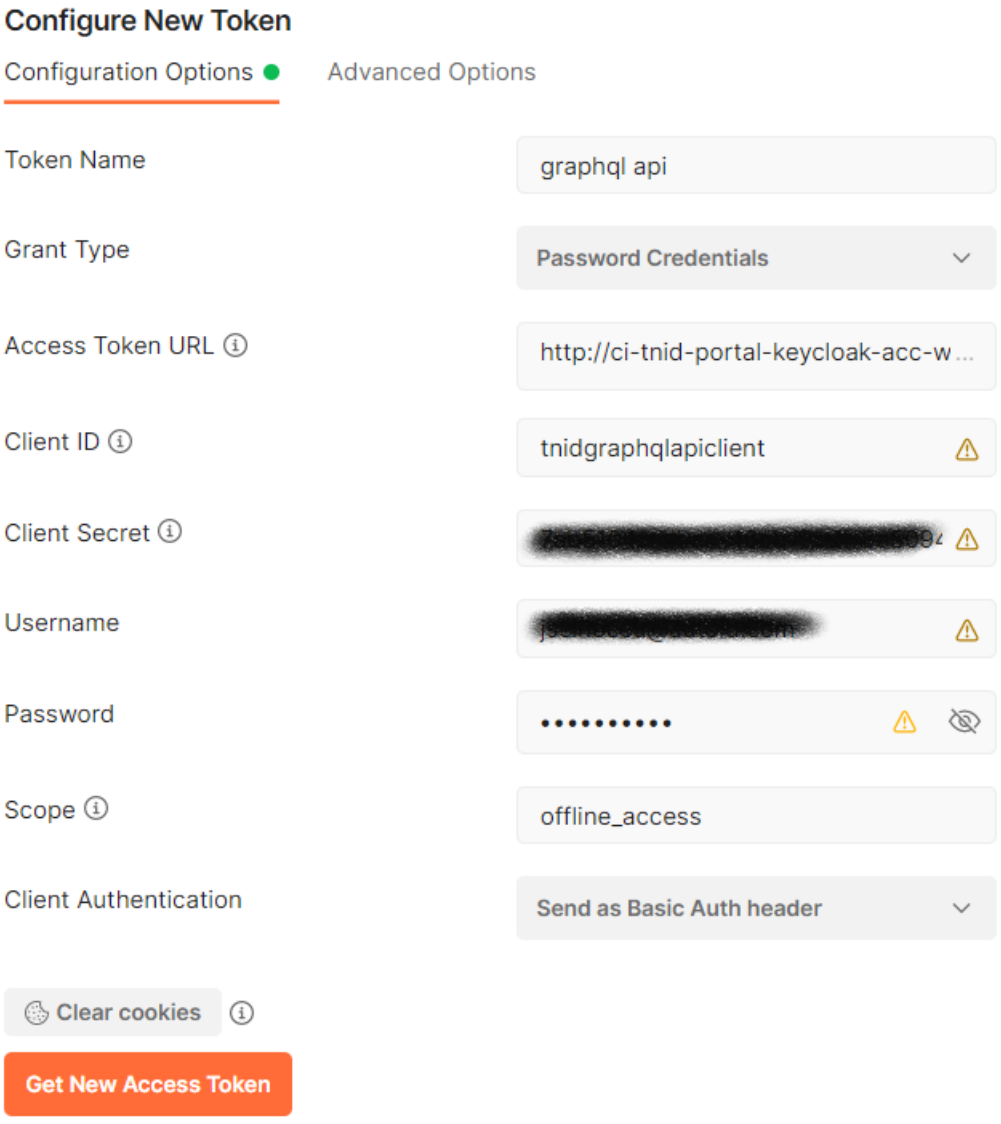
Resolution:
Check if the client id is correct.

Postman Examples

Postman (https://www.postman.com) is an API platform for building and using API's. It is a tool that can be used to execute queries against various API's, like REST and GraphQL.

Authorization Token

Every POST to the TN.ID endpoint must contain an Authentication Header that contains a Bearer Token. To request a token, use the following data (Collection -> Authentication):



For Access Token URL, enter the url [Token_Url]. The *Scope* field is optional.

After clicking *Get new Access Token*, Postman should respond with

Authentication complete

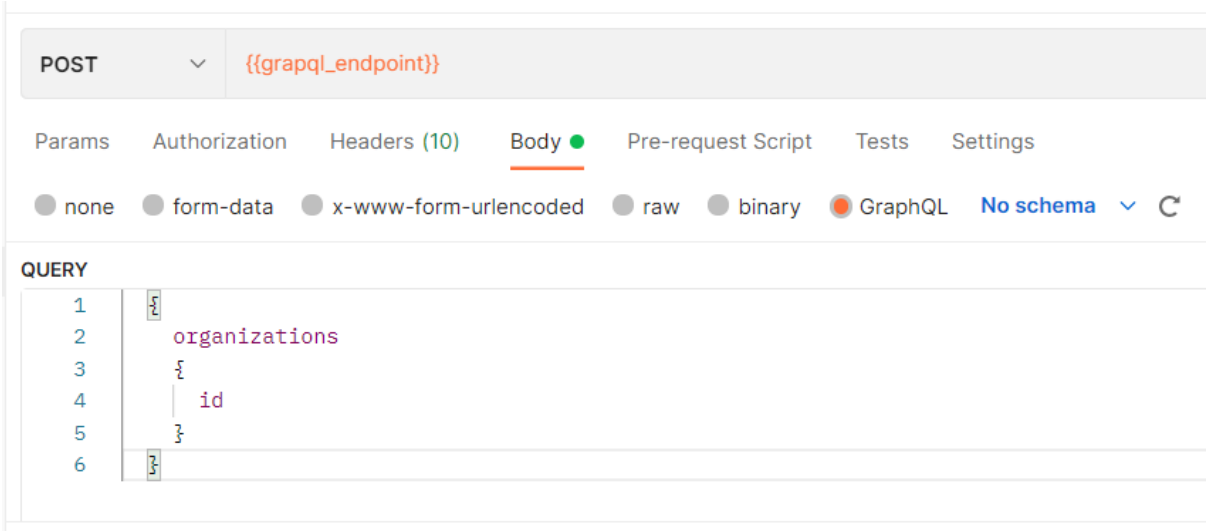
In the next dialog, the *Access Token* and *Refresh Token* are displayed.

Token Details **Use Token**

Token Name	graphql api
Access Token	eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJPaHBHT1VP TWNaUVp5WUw3TzNFZlhlbTNueVFmMHZBSndLZF9JSGtlUDhBIn 0.eyJleHAiOjE2NDMyODk4ODEsImlhdCI6MTY0MzI4OTU4MSwian RpIjoiYzViYjllYWUtMTQyNS00OTg3LTk3YmMtZDE0NzJiOGUzMGE
Token Type	Bearer
expires_in	300
refresh_expires_in	0
refresh_token	eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIyNGFlYzRkMy 1jMDVmLTRhNzctOGU5MS04NzcyNTZkZTY4YjlifQ.eyJpYXQiOjE2N DMyODk1ODEsImp0aSI6ImY3NDcwYTdlLTUyMzQtNGE4NC1hNDZj

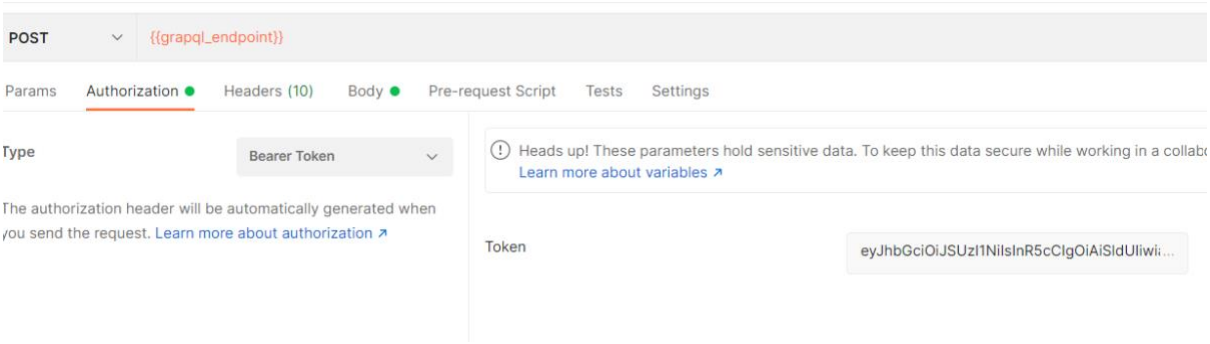
Request Organization(s)

In Postman, create a new Request and make sure that in the **Body** tab **GrapQL** is selected:

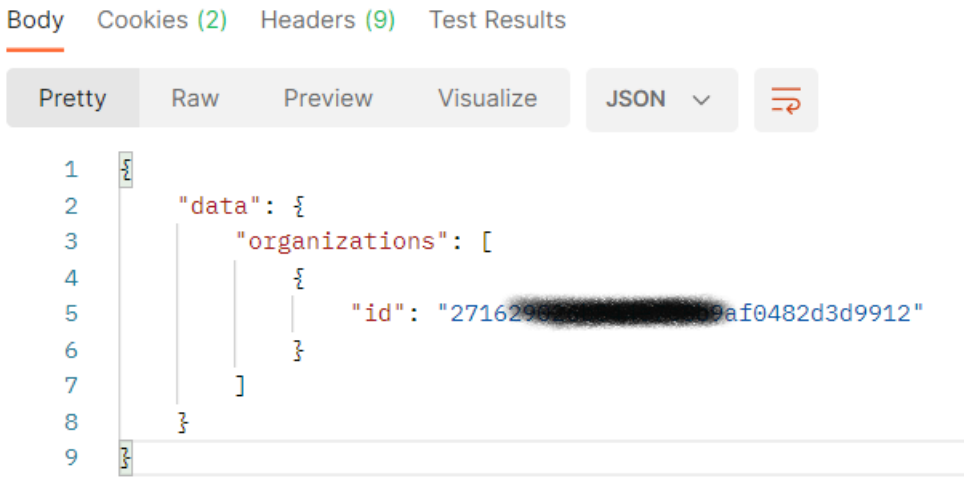


For {{graphql_endpoint}}, use the value for [GraphQL_Endpoint].

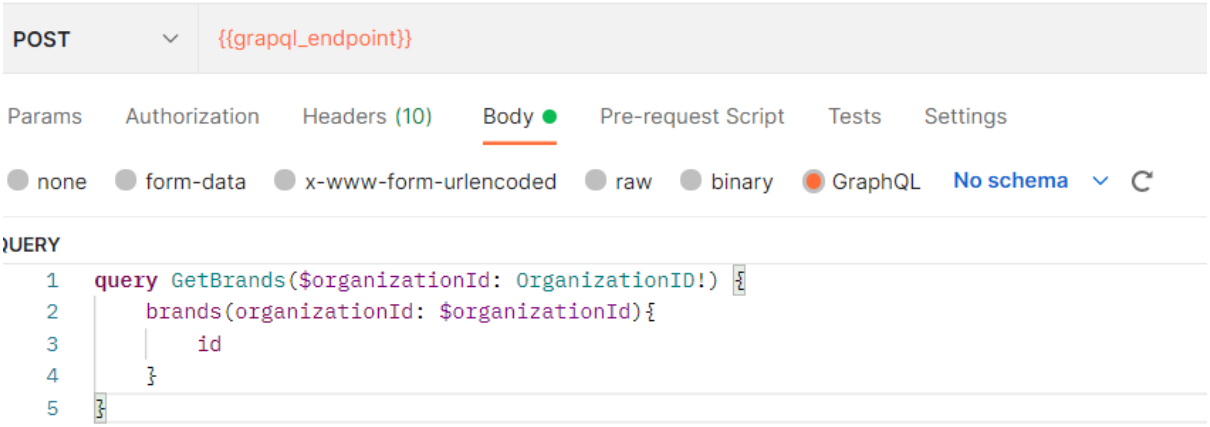
On the **Authorization** tab, choose **Bearer Token** and enter the token received earlier:



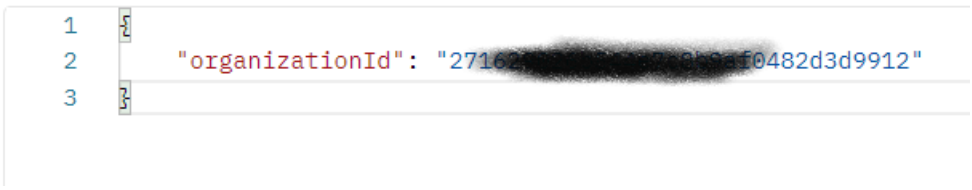
When clicking **Send**, the response should return the *organization id* that the user has access to:



This id can be used for subsequent calls for this organization to the TN.ID GraphQL endpoint, for example, get a list of Brands:

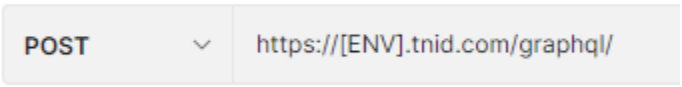


Using the organizationId as a GraphQL variable:



Create a Brand

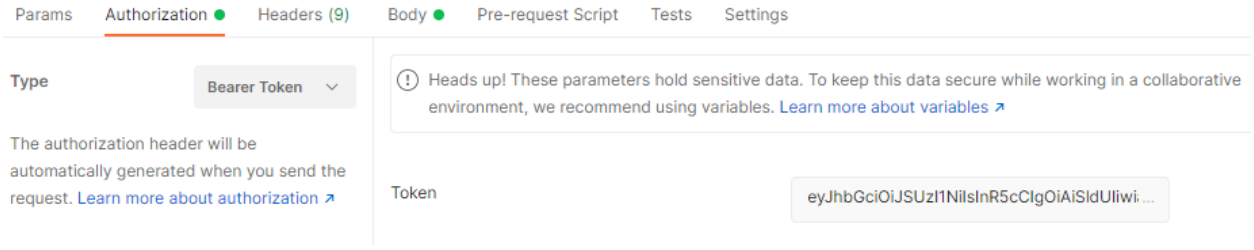
Step 1: Set endpoint



Set the Method to **POST** and the url to the correct url for the environment.

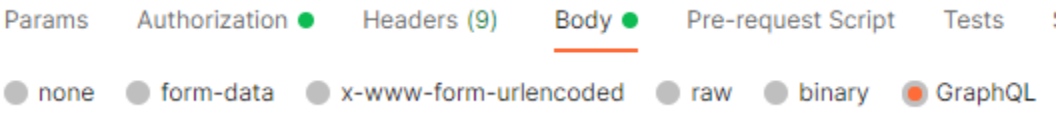
Step 2: Set authorization

In the tab **Authorization** set the **Type** to *Bearer Token* and enter the token that is acquired from the token endpoint in the **Token** field:



Step 3: Set Body

In the tab **Body**, make sure that the body type is set to **GrapQL**:



In the **QUERY** field, enter the mutation to create a Brand:

```
QUERY
1  mutation createBrand ($input: CreateBrandInput) {
2      createBrand (input: $input) {
3          id
4      }
5  }
```

The fields that you want to receive from the created **Brand** can be listed immediately from the mutation. In this case, only the *id* field is requested.

In the GRAPHQL VARIABLES field, enter the variables in the following format, so that the *$input* variable is correctly mapped to the *"input"* variable field:

GRAPHQL VARIABLES

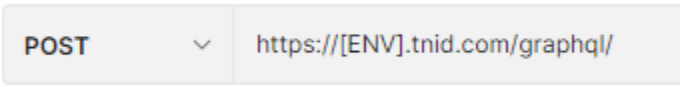
```
{
    "input":
    {
        "organizationId": "89a05c67-1fd7-4330-8fd9-ddf9d780dc95",
        "entityType": "SOLE_PROPRIETOR",
        "firstName": "First",
        "lastName": "Last",
        "displayName": "First Last",
        "companyName": "My Company Name",
        "commercialContactPhoneNumber": "+19999999999",
        "technicalContactPhoneNumber": "+19999999999",
        "financialContactPhoneNumber": "+19999999999",
        "street": "1 E Kennedy Blvd",
        "city": "City",
        "state": "WA",
        "postalCode": "12345",
        "country": "US",
        "technicalContactEmail": "no-reply@email.com",
        "commercialContactEmail": "no-reply@email.com",
        "financialContactEmail": "no-reply@email.com",
        "brandRelationship": "BASIC_ACCOUNT",
        "vertical": "REAL_ESTATE"
    }
}
```

Step 4: Send request

Click **Send** and wait for a response. If the response is successful, the id of the created Brand is returned:

Create a Campaign

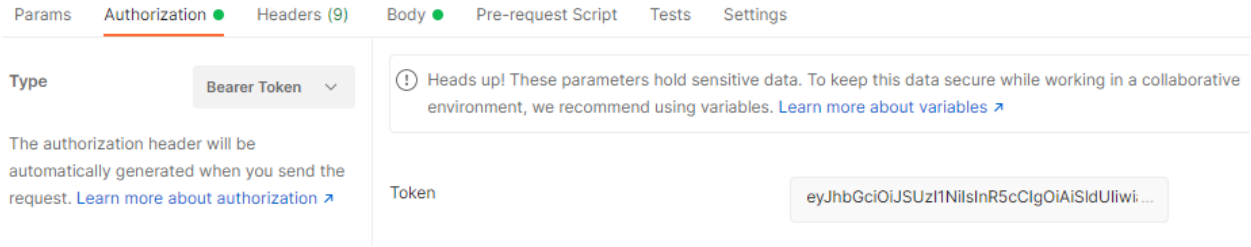
Step 1: Set endpoint



Set the Method to POST and the url to the correct url for the environment.

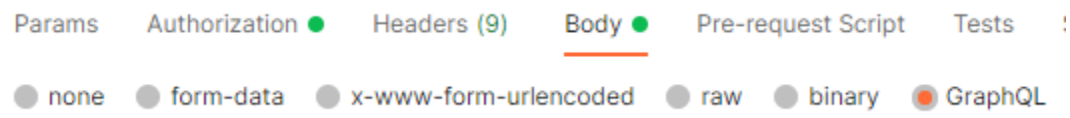
Step 2: Set authorization

In the tab **Authorization** set the **Type** to *Bearer Token* and enter the token that is acquired from the token endpoint in the **Token** field:



Step 3: Set Body

In the tab **Body**, make sure that the body type is set to **GrapQL**:



In the **QUERY** field, enter the mutation to create a Campaign:

```
QUERY
1  mutation CreateCampaign($input: CreateCampaignInput!) {
2    createCampaign(input: $input) {
3      id
4    }
5  }
```

The fields that you want to receive from the created **Campaign** can be listed immediately from the mutation. In this case, only the *id* field is requested.

In the GRAPHQL VARIABLES field, enter the variables in the following format, so that the *$input* variable is correctly mapped to the *"input"* variable field:

GRAPHQL VARIABLES

```
{
    "input": {
        "brandId": "98bc21d61f364437b1169cc576011825",
        "vertical": "TECHNOLOGY",
        "description": "Basic marketing.",
        "sampleMessages": [ "Sample1", "Sample2" ],
        "usecase": "SOLE_PROPRIETOR",
        "subUsecases": [ "MARKETING" ],
        "affiliateMarketing": false,
        "ageGated": false,
        "directLending": false,
        "subscriberHelp": true,
        "embeddedLink": false,
        "embeddedPhone": false,
        "numberPool": false,
        "subscriberOptin": true,
        "subscriberOptout": true,
        "phoneNumbers": [ "19999999999" ],
        "autoRenewal": false,
        "helpMessage": null,
        "messageFlow": null,
        "referenceId": null
    }
}
```

Step 4: Send request

Click **Send** and wait for a response. If the response is successful, the id of the created Campaign is returned:

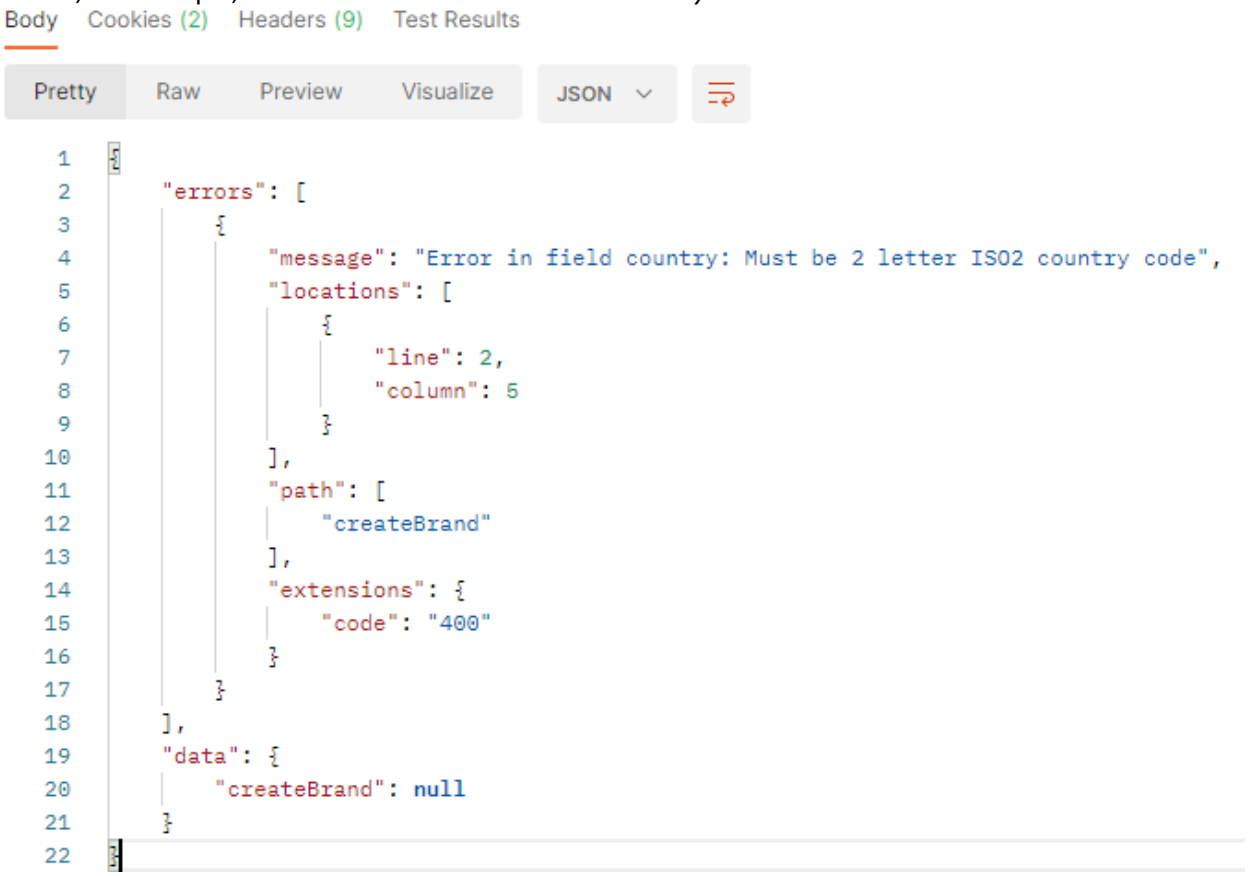
Body Cookies (2) Headers (11) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "data": {
        "createCampaign": {
            "id": "af8c66[redacted]cba8b78b44b6f76e1"
        }
    }
}
```

Validation errors

The input data is validated and if any validation errors are found, an error object is returned with the details, for example, when an invalied value for the *Country* field is entered:




```
{
    "errors": [
        {
            "message": "Error in field country: Must be 2 letter ISO2 country code",
            "locations": [
                {
                    "line": 2,
                    "column": 5
                }
            ],
            "path": [
                "createBrand"
            ],
            "extensions": {
                "code": "400"
            }
        }
    ],
    "data": {
        "createBrand": null
    }
}
```

cURL

The following examples have been setup using curl in a Windows command prompt. Note that curl needs other settings when used otherwise. For example, the continuation-on-next-line character is ^ when using cmd.exe, but ` when using powershell, or \ when using linux. Also, the use of quote characters (single or double) may differ.

Authentication token

To retrieve an authentication token, use the following command, while substituting the values from the variable table:

Request

```
curl -X POST [Token_Url] ^
--data-urlencode "grant_type=password" ^
--data-urlencode "client_id=[Client_ID]" ^
--data-urlencode "client_secret=[Client_Secret]" ^
--data-urlencode "user_name=[User_Name]" ^
--data-urlencode "user_name=[Password]"
```

Response

The response is in the form of a Json-formatted string:


```
{
  "access_token":"eyJ<...>XI_Q",
  "expires_in":300,
  "refresh_expires_in":1800,
  "refresh_token":"eyJ<...>f4s",
  "token_type":"Bearer",
  "not-before-policy":0,
  "session_state":"da3bda59-6840-43bb-b641-577088f922ca",
  "scope":"profile organizations email"
}
```


Authentication token using a refresh token

With the refresh token a new access token can be retrieved, without using username and password:

Request

```
curl -X POST [Token_Url] ^
--data-urlencode "grant_type=refresh_token" ^
--data-urlencode "client_id=tnidgraphqlapiclient" ^
--data-urlencode "client_secret=[Client_Secret]" ^
--data-urlencode "refresh_token=eyJ<...>f4s"
```

Response


```
{
  "access_token":"ey<…>86lk",
  "token_type":"Bearer",
  "not-before-policy":0,
  "session_state":
  "d9282a42-8b9a-4a15-97e5-5fa9a40cfd2a",
  "scope":"profile organizations email offline_access"
}
```


Query the GraphQL Endpoint

With the access token, call the GraphQL endpoint using the following command:

Request

```
curl -X POST "[GraphQL_Endpoint]" ^
--header "Authorization: Bearer ey<…> v9sA" ^
--header "Content-Type: application/json" ^
--data-raw "{\"query\":\"<QUERY>\",\"variables\":{}}"
```

Mutate the GraphQL Endpoint

Use the following command structure:

Request

curl -X POST "[GraphQL_Endpoint]" ^
--header "Authorization: Bearer ey<…> v9sA" ^
--header "Content-Type: application/json" ^
--data-raw "{\"query\":\"mutation mutationName ($input: MutationInput) { mutationName (input: $input) { id } }\",\"variables\":{\"input\":{ \"var1\":\"value1\" }}}"

Formatting of Queries and Mutations

The JSON-format of the query and mutation is as follows:

```
{
  "query":"<Query>",
  "variables":<Variables>
}
```

Note that a mutation is also a query. A mutation operation is identified within the query. A query-only is formatted as:

```
{
  "query": "{ entity { field1, field2 } }",
  "variables": <variables>
}
```

A mutation is formatted as:

```
{
  "query": "mutation myMutation ($input: EntityInput)
       {
             myMutation (input: $input) { field1, field2 }
       }"
}
```

Request Organization(s)

The GraphQL query for requesting an organization is:

```
{
  organizations {
    id
  }
}
```

The formatted query with the escaped double quotes is then:

```
curl -X POST "[GraphQL_Endpoint]" ^
--header "Authorization: Bearer ey<…> v9sA" ^
--header "Content-Type: application/json" ^
--data-raw "{\"query\":\"{ organizations { id } }\",\"variables\":{}}"
```

The response is a Json-formatted string:

```
{
       "data": {
             "organizations": [
```

```
                {
                        "id": "8e91475dfd664c7eb2b512accaaf7752"
                }
            ]
        }
}
```

Create Brand

To create a brand, use the following mutation and variables:

Mutation:

```
mutation createBrand ($input: CreateBrandInput) {
 createBrand (input: $input) {
   id
 }
}
```

Variables:

```
{
  "input": {
    "organizationId": "61bd7af075fe4e7b830a4b74adda8b78",
    "entityType": "SOLE_PROPRIETOR",
    <..SKIPPED..>
    "vertical": "REAL_ESTATE"
  }
}
```

Note: All double quotes must be escaped using a backslash '\' character.

The formatted request including variables is then:

```
curl -X POST "[GRAPHQL_Endpoint]" ^
--header "Authorization: Bearer eyJh<…>E6Yg" ^
--header "Content-Type: application/json" ^
--data-raw "{ \"query\":\"mutation createBrand ($input: CreateBrandInput)
{ createBrand (input: $input) { id }}\", \"variables\": { \"input\":
{ \"organizationId\": \"00883676c13c4992aa46225c0f3036d9\", \"entityType\":
\"SOLE_PROPRIETOR\", \"firstName\": \"First\", \"lastName\": \"Last\",
\"displayName\": \"First Last\", \"companyName\": \"My Company Name\",
\"commercialContactPhoneNumber\": \"+1999999999999\",
\"technicalContactPhoneNumber\": \"+1999999999999\", \"financialContactPhoneNumber\":
\"+1999999999999\", \"street\": \"1 E Kennedy Blvd\", \"city\": \"City\", \"state\":
\"WA\", \"postalCode\": \"12345\", \"country\": \"US\", \"technicalContactEmail\":
\"no-reply@email.com\", \"commercialContactEmail\": \"no-reply@email.com\",
\"financialContactEmail\": \"no-reply@email.com\", \"brandRelationship\":
\"BASIC_ACCOUNT\", \"vertical\": \"REAL_ESTATE\" } } }"
```

The response is a json-formatted string, containing the requested fields (in this case: only *id*) from the created Brand:

```
{"data":{"createBrand":{"id":"9bc7c8346e484f1c9e7fb8d8ed559f03"}}}
```

Create Campaign

Following the previous example, the same formatting can be used to create a Campaign:

Mutation:

```
mutation createCampaign ($input: CreateCampaignInput) {
 createCampaign (input: $input) {
   id
 }
}
```

Variables:

```
{
  "input": {
    "brandId": "61bd7af075fe4e7b830a4b74adda8b78",
    "vertical": "TECHNOLOGY",
    <..SKIPPED..>
    "description": "Basic"
  }
}
```

Note: All double quotes must be escaped using a backslash '\' character.

The formatted request including variables is then:

```
curl -X POST "https://acc.tnid.com/graphql" ^
--header "Authorization: Bearer eyJh<…>4KzSA" ^
--header "Content-Type: application/json" ^
--data-raw "{ \"query\":\"mutation createCampaign ($input: CreateCampaignInput)
{ createCampaign (input: $input) { id }}\", \"variables\": { \"input\":
{ \"brandId\": \"9bc7583463714f6c9e7fbfd8ede59f03\", \"vertical\": \"TECHNOLOGY\",
\"description\": \"Basic marketing.\", \"sampleMessages\": [ \"Sample1\",
\"Sample2\" ], \"usecase\": \"SOLE_PROPRIETOR\", \"subUsecases\": [ \"MARKETING\" ],
\"affiliateMarketing\": false, \"ageGated\": false, \"directLending\": false,
\"subscriberHelp\": true, \"embeddedLink\": false, \"embeddedPhone\": false,
\"numberPool\": false, \"subscriberOptin\": true, \"subscriberOptout\": true,
\"phoneNumbers\": [ \"19999999999\" ], \"autoRenewal\": false, \"helpMessage\": null,
\"messageFlow\": null, \"referenceId\": null } } }"
```

The response is a json-formatted string, containing the requested fields (in this case: only *id*) from the created Campaign:

```
{"data":{"createCampaign":{"id":"c893923965963d37a6abfff33a02de9c"}}}
```

.NET examples

These examples are created with help of the nuget package RestSharp (https://restsharp.dev/)

Authentication token

To receive an authentication token

```
using RestSharp;

var clientId = "[Client_ID]";
var clientSecret = "[Client_Secret]";
var userName = "[User_Name]";
var password = "[Password]";
var tokenEndpoint = "[Token_Url]";

var client = new RestClient(tokenEndpoint);
var request = new RestRequest();

request.AddParameter("grant_type", "password");
request.AddParameter("client_id", clientId);
request.AddParameter("client_secret", clientSecret);
request.AddParameter("username", userName);
request.AddParameter("password", password);

var response = await client.ExecutePostAsync(request);

Console.WriteLine(response.Content);
```

Authentication token using a refresh token

With the refresh token a new access token can be retrieved, without using username and password:

```
using RestSharp;

var clientId = "[Client_ID]";
var clientSecret = "[Client_Secret]";
var tokenEndpoint = "[Token_Url]";
var refreshToken = "[YOUR REFRESH TOKEN]";

var client = new RestClient(tokenEndpoint);
var request = new RestRequest();

request.AddParameter("grant_type", "refresh_token");
request.AddParameter("client_id", clientId);
request.AddParameter("client_secret", clientSecret);
request.AddParameter("refresh_token", refreshToken);

var response = await client.ExecutePostAsync(request);

Console.WriteLine(response.Content);
```

Query the GraphQL Endpoint

```
using RestSharp;

const string ApplicationJson = "application/json";

var authToken = "YOUR AUTH TOKEN";
var endpoint = "[GraphQL_Endpoint]";

var graphQlQuery = "{ \"query\": \"{ organizations { id } }\" }, \"variables\":
\"{}\"";

var client = new RestClient(endpoint);
var request = new RestRequest();

request.AddHeader("Authorization", $"Bearer {authToken}");
request.AddHeader("Content-Type", ApplicationJson);
request.AddBody(graphQlQuery, ApplicationJson);

var response = await client.ExecutePostAsync(request);

Console.WriteLine(response.Content);
```

Mutate the GraphQL Endpoint

Use the following command structure:

```
using RestSharp;

const string ApplicationJson = "application/json";

var authToken = "YOUR AUTH TOKEN";
var endpoint = "GRAPHQL_ENDPOINT";

var graphQlMutation = "{ \"query\":\"mutation mutationName ($input:
MutationInput) { mutationName (input: $input) { id } } \", \"variables\": {
\"input\": { \"var1\":\"value1\" }} }";

var client = new RestClient(endpoint);
var request = new RestRequest();
request.AddHeader("Authorization", $"Bearer {authToken}");
request.AddHeader("Content-Type", ApplicationJson);
request.AddBody(graphQlMutation, ApplicationJson);

var response = await client.ExecutePostAsync(request);

Console.WriteLine(response.Content);
```

Classes and Serialization

For serializing classes to the json format, either *System.Text.Json* (since .net 4.8 or netstandard 2.0) or the nuget package Newtonsoft.Json can be used. For example, the following simple class structure can be used for creating a more robust request:

```
namespace TNIDSamples
{
    public class QueryRequest
    {
        public string Query { get; set; }

        public Variables Variables { get; set; }
    }

    public class Variables
    {
        public Input Input { get; set; }
    }

    public class Input
    {
        public Guid OrganizationId { get; set; }

        // Insert other fields here
    }
}
```

These classes can be serialized to create a mutation:

```
using RestSharp;
using System.Text.Json;
using TNIDSamples;

const string ApplicationJson = "application/json";

var authToken = "YOUR AUTH TOKEN";
var endpoint = "GRAPHQL_ENDPOINT";

var graphQLRequest = new QueryRequest
{
    Query = "mutation createBrand ($input = CreateBrandInput) { createBrand
(input: $input) { id }}",
    Variables = new Variables
    {
        Input = new Input
        {
            OrganizationId = Guid.NewGuid()
        }
    }
};

var jsonOptions = new JsonSerializerOptions { PropertyNamingPolicy =
JsonNamingPolicy.CamelCase };
var json = JsonSerializer.Serialize(graphQLRequest, jsonOptions);

var client = new RestClient(endpoint);
var request = new RestRequest();
request.AddHeader("Authorization", $"Bearer {authToken}");
request.AddHeader("Content-Type", ApplicationJson);
request.AddBody(json, ApplicationJson);

var response = await client.ExecutePostAsync(request);

Console.WriteLine(response.Content);

Console.ReadLine();
```

Example 10DLC Registration Workflow

Below is a basic step-by-step process of registering your organization, creating brands and campaigns, and mapping numbers for approved messaging in the 10DLC ecosystem:

1) Register Your Organization
    a. Get an API key by emailing developers@tsgglobal.com and providing your organization details (including legal business name, address, and tax ID number).
2) Register Brands
    a. Create Brands for either your own Organization (if you are sending outbound messages where the end-user believes your company is the sender) or create Brands for your customers/downstream clients.
        i. This can be done in the GUI or via the createBrand mutation.

3) Create Campaigns
    a. Create Campaigns for your Brands using the GUI or the createCampaign mutation.
    b. Standard use case campaigns are approved immediately for qualified Brands. Generally, some special use campaigns are dependent on MNO review times.
    c. You can associate numbers to the Campaigns in the createCampaign mutation, or at the end of the wizard in the UI.
    d. Once the Campaign is approved, numbers are mapped to the Campaign, and TNID will forward any relevant information to the required databases.

Additional Features

Below are features that are available in TNID to assist with number and campaign management:

Tagging

Users can add multiple unique tags to both campaigns and to brands upon creation. You are also able to add tags post-creation. Tags can be used to search for specific campaigns or brands based on keywords you may find useful. You can filter by tags on the primary Campaign and Brand pages.

Logos

Users can upload image logos to Partners or Brands, allowing for faster recognition of Partners or Brands.

3rd Party Services Supported By TNID

At this time, TNID currently manages data with the below industry registries, databases, or services:

Service or Registry	Description	More Information
The Campaign Registry	10DLC management	https://www.campaignregistry.com/
NetNumber Override Service Registry (OSR)	10DLC management and NNID information	https://www.netnumber.com/
Syniverse	10DLC management and forwarding	https://www.syniverse.com

	of campaigns to T-Mobile/Sprint	

This list will be updated as we integrate with additional services.

Security

No VPNs or other special network configurations are required at this time to access the TNID portal or GraphQL APIs.

SLA & Performance

There are currently no imposed API limits. Please refer to your beta agreement for more information.

Support

Email us at: developers@tsgglobal.com