Reputation API

Legal

This document is confidential and may not be copied, shared or redistributed without the explicit permission of Token of Trust, LLC.

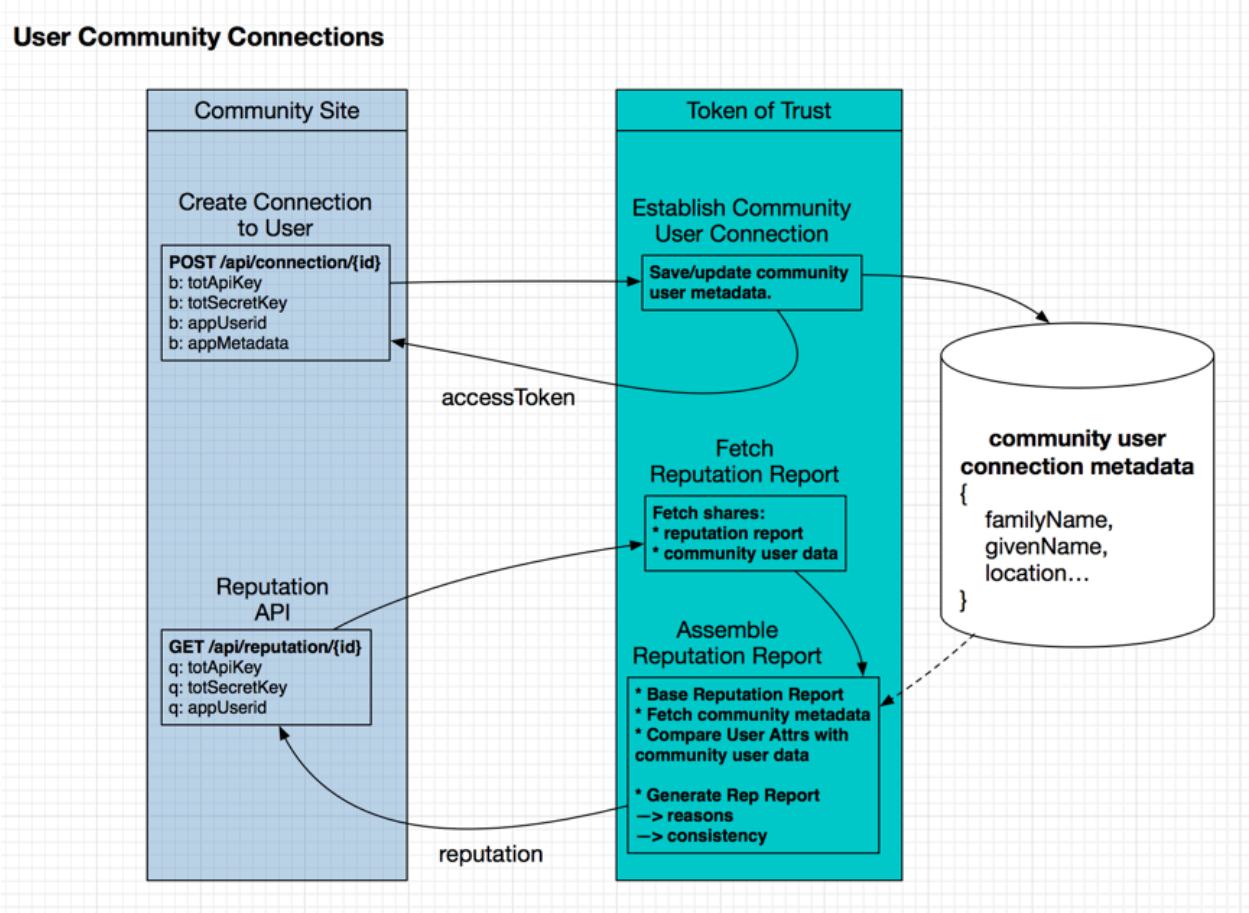
STATUS: The Reputation API is largely obsoleted except for some very narrow use cases. Most customers will want to use the Verify Person API mentioned below since it is easier to use, returns an access token and creates a connection for you.

Token of Trust's public API documents are available publicly here:

- **Verify Person API**
- **Reputation API**

. This documentation should be updated to refer to the Verify Person API.

Overview: How Connection and Reputation APIs work Together



When the `/reputation` endpoint is invoked it pulls the end-user's baseline reputation report and compares the metadata Token of Trust has assembled for the user with metadata from the community to assemble a final Reputation Report. The community may change the metadata stored for that user at any time by calling the `/connection` endpoint at any time.

Changes to the connection metadata affect the end-user's widgets and reputation api immediately. If an overall 'noMatch' response is the result of the compare between the connection metadata and the user Token of Trust attributes the user's Reputation Summary will display 'Not Verified' and the Verified Indicator will be hidden.

Communities should set location, given name, family name, and date of birth to make best efforts to ensure consistency between community data and the connected Token of Trust account. At minimum if these values are displayed they should be set so that clear inconsistencies between the community and Token of Trust are captured as soon as possible.

Scopes and Permissions

Permissions define what an application is allowed to access. Permissions are determined based upon your current application subscription level and will change if/when you change your subscription.

Scope defines what an API request should return and may be passed along with every request. If you don't have the permission to use a given scope will be removed or the API request rejected. Here are the permissions - each of which also defines a scope:

Permission	**Description**
seal	Basic reasons and consistency related to the end user (sections below for details).
seal-details	'seal' access plus dependsOn (sections below for details).

API Basic Structure

All APIs return responses of this form:

```
{
      "content" {
            /** varies depending upon API Response **/
      },
      "metadata" {
            "version":"1.0.0",
            "status":200
      }
}
```

GET /api/reputation/{appUserid}

Allows access to elements of the user's reputation report. Specifically to the reasons information.

Request Spec

```
'spec': {
      method: 'GET',
      path: '/api/reputation/:appUserid',
      description: 'Returns the reputation report for the specified user.',
```

```
      produces: ['application/json'],
      parameters: [
            param.path('appUserid', 'community ID of user', 'string'),
            param.query('totApiKey', 'A valid API key.', 'string'),
            param.query('totSecretKey', 'A secret key..', 'string'),
            param.query('appDomain', 'The domain registered for this api key.',
'string'),
            param.query('scope', 'Controls the type and amount of data to
return - the app must have permissions to this scope as provided by end users
and/or payment plan. Optional - but we recommend including this field since
more/less data may be returned as the api is versioned and default scoping
changes.', 'string')],
}
```

Parameters

You've seen most of the parameters mentioned above… the one exception is 'scope':

Scope

Controls the type and amount of data to return. Not all applications can get to all user data. The data returned in the 'reputation' object can vary based upon the permissions users have granted the app and the level of API the application is subscribed to. If it is not specified more/less data may be returned as the api is versioned and default scoping changes. Passing scope is optional - but highly encouraged because it stabilizes the response from Token of Trust. To get the fastest response - please ask for only the data you're likely to need.

See Scopes and Permissions for more details.

Response

Assuming the user is found - at minimum the following fields are included in the 'content' section of the response:

```
            connection: /* Connection Status Object (see below) */,
            reasons: /* map of Reason Objects (see below) */,
            consistency: /* map of Consistency Objects (see below) */,
            privateDataGranted: /* map of data the consumer has granted
permission to share (see below) */
            testMode: true,
```

Connection Status Object

The Connection Status object reports the state of the connection for this user.

```
      connection : {
            status : 'active' | 'restricted' | 'noUser',
            appUserid : 'YOUR_APPUSERID',
```

```
        reasonsForRestriction : [
          {
            "type": "rightToRestrict",
            "initiator": "self",
            "id": "KCb95l",
            "createdTimestamp": "2018-05-18T16:04:24.608Z",
            "updatedTimestamp": "2018-05-18T16:04:24.608Z"
          }
        ]
    }
```

Here's a brief description of what these attributes mean:

Attribute	Definition
`status`	The current state of the connection. `'active'` means an active user that is connected to your network. 'restricted' is a valid user that has deactivated their account for one or more reasons (see `'reasonsForRestriction'`). `'noUser'` means that this appUserid has not yet taken the first step to connect to Token of Trust.
`appUserid`	same as the `appUserid` you passed in and that this record relates to.
`reasonsForRestriction`	Only present if status the status is `'restricted'`. Gives a sense of why the restriction is in place via one or more reasons with the attributes below.
`initiator`	When 'initiator' is 'self' user has opted to restrict their profile but it can be restricted by `'totSupport'` or `'totSystem'` as well - generally for malicious activity or because the account is `'pendingErasure'`.
`type`	The type of reason for the restriction. Valid values are `'rightToRestrict'`, `'pendingErasure'`, `'reportOfAbuse'`.
`createdTimestamp` `updatedTimestamp`	Timestamps to indicate when the restriction was placed or updated.
`id`	A unique identifier for the reason.

Reason Objects

The `reasons` attribute is an object that has a map of Reasons Objects (see the Example below). There are some common structural elements to be aware of.

The first is that all reasons are key-value pairs that follow this form

```
      someReason : {
          value : true | false | "noMatch" | "partialMatch" | "fullMatch" |
"insufficientData",
             ---
          otherKeys : otherAttributes
      }
```

In short - every reason has a specific key (above) and a 'value' attribute that has the following meaning:

value	**Definition**
`true`	same as `partialMatch` (see below).
`false`	same as `noMatch` (see below).
`noMatch`	this reason is not true or otherwise does not pass the minimum criteria to satisfy this reason.
`partialMatch`	this reason meets the minimum criteria but does not meet the fullMatch criteria to satisfy this reason.
`fullMatch`	this reason meets the full match criteria and is the best possible match currently defined by Token of Trust.
`insufficientData`	there was not enough data to rationalize about this reason. This is result is indeterminate - neither true, nor false.

The difference and boundaries between no, full and partial matches depends upon the reason. Other attributes may also accompany each reason - see the reason description for more details.

Example Reason Object

Items with an (*) are returned for **seal** scope - other items are returned only for **seal-details** scope. ‘dependsOn’ are only provided for seal-details API:

```
reasons: {
      "*isUserBaselineVerified": {
        "value": false,
        "dependsOn": [
          "isOnlineVerified", “accountDataIsConsistent”
        ]
      },
      "*isRealWorldVerified": {
        "value": false
      },
      "accountDataIsConsistent": {
        "value": "fullMatch",
        "dependsOn": [
          "locationIsConsistent",
          "givenNameIsConsistent",
          "familyNameIsConsistent"
        ]
      },
      "familyNameIsConsistent": {
        "compared": 2,
        "value": "fullMatch",
        "fullMatches": 2
      },
      "locationIsConsistent": {
        "compared": 2,
        "value": "fullMatch",
        "fullMatches": 2
      },
      "locationLocalityRegionIsConsistent": {
        "compared": 2,
        "value": "fullMatch",
        "fullMatches": 2
      },
      "givenNameIsConsistent": {
        "compared": 2,
        "value": "fullMatch",
        "fullMatches": 2
      },
      "*isOnlineVerified": {
        "value": false,
        "dependsOn": [
          "facebook_totVerifiedAccount",
          "paypal_totVerifiedAccount"
        ]
```

```
    }
    "isOnlineVerified": {
        "dependsOn": [
          "accountDataIsConsistent",
          "facebook_totVerifiedAccount",
          "paypal_totVerifiedAccount"
        ],
        "value": "noMatch"
    },
    "facebook_totVerifiedAccount": {
        "value": "noMatch"
   }
}
```

Example Consistency Objects

The intent here is to ensure that we can show that a user who opts-in has demonstrated some level of consistency with the data that the community has on hand related to that user - without compromising that user's privacy. Consistency for each dimension is per the Matching section below.

For some use cases it is important to note that the quality of the data could be wildly different. In general government ID and other sources bound to the real world are considered more reliable. The **seal-details** scope provides a breakdown of the number of compares and matches for online vs real world verifications.

Only the 'status' attributes are returned for **seal** scope. 'status', 'matches', 'partialMatches' and 'compares' are returned for **seal-details** scope:

```
givenName : {
      status: 'fullMatch|partialMatch|noMatch',
      compares : {realWorld:1, online:2, primary: 1},
      fullMatches: {online:1, primary: 1},
      partialMatches: {realWorld:1, online:1}
},
familyName : {
      status: 'fullMatch|partialMatch|noMatch',
      compares : {realWorld:1, online:2, primary: 1},
      fullMatches: {realWorld:1, online:1},
      partialMatches: {online:1, primary: 1}
},
middleName : {
      status: 'fullMatch|partialMatch|noMatch',
      fullMatches: {realWorld:1},
      partialMatches: {online:2, primary: 1}
}
```

The number of `fullMatches` and `partialMatches` will be returned for each attribute - how items are matched up depend upon the attribute in question. See Matching below).

Location Activities

Token of Trust will deliver consistency information related to to a user's activities within the user's location consistency objects using the variables outlined below:

"`activities`" - Reports a summary of fact details related to location information that we've ascertained from the user's online activities. These activities include:

> "`mostRecentlySeenTimestamp`" - a timestamp of the most recent activity near the primary location.
> "`earliestSeenTimestamp`" - a timestamp of the earliest activity near the primary location.
> "`distinctSightings`" - the number of occurrences where a user was tagged within an activity that contained location information (more detail below).
> "`fullMatches`" - the number of occurrences where a user's location activity reported them within the smaller radial range of the primary location (recommending <= 5 miles).
> "`partialMatches`"- the number of occurrences where a user's location activity reported them within the larger radial range of the primary location (recommending > 5 miles and < 15 miles).

distinctSightings

Distinct sightings are drawn from the most recent 10,000 photos where each sighting must have:

- A tagged location with GPS coordinates
- At least one friend has interacted with it (comment or like)
- The photo's data was not altered (back-dated, location-changes, etc)

From these `fullMatches` are those within the `'maxFullMatch'` distances and `partialSightings` are those within the `maxPartialMatch` distance. A `fullMatch` is never double counted in `partialMatch`es.

privateDataGranted

Private data the consumer provided Token of Trust is returned on the reputation api when the consumer has granted permission to share that information. This can be information the consumer provided directly or information collected from documents shared with Token of Trust. This information is only shared if the consumer has granted permission.

The data is returned in the following format:

```
"privateDataGranted": {
      "govtId.givenName": "TOM",
      "govtId.familyName": "JONES"
  },
```

The exact list of attributes will depend upon what was permitted and supplied by the user. Consent is generally collected by way of the Verify Person API.

Matching

Matching is done across all attributes and verification points to ensure consistency of TOT and community data.

Here is how we assess `fullMatch`, `partialMatch` and `noMatch` indications for each attribute:

Attribute	**fullMatch**	**partialMatch**	**noMatch**
Location	All locations within 5 km radius.	All locations within 10 mile radius.	At least one location beyond 20 km radius.
Given Name	If a string - an exact match of the string. Otherwise all names in ToT found in dictionary.	At least one match name, but not all names match.	No match on name.
Middle Name	If a string - an exact match of the string. Otherwise all names in ToT found in dictionary.	At least one match name, but not all names match.	No match on name.
Family Name	If a string - an exact match of the string. Otherwise all names in ToT found in dictionary.	At least one match name, but not all names match.	No match on name.

Name Matching

Name matching is complex. People have first (given) names, last (family) names, nicknames, maiden names, past names and aliases that they choose to use to protect their anonymity when interacting online. In parts of the world people use maternal and paternal names and in other parts people hyphenate their names when they get married.

Not all of these uses evoke the same feelings of trust but they are all cases that people use. Token of Trust attempts to handle all of these cases. To understand more about the full scope of what is possible with names and Token of Trust please (see Specifying Names).

Handling familyName

```
"familyName": {
  current: "Fernández",
  previous: "Fernández",
  maternal: "Fernández",
  paternal: "Martínez"
}
```

Handling givenName

Example Curl Request for /api/reputation

```
curl -X GET \
"https://app.tokenoftrust.com/api/reputation/YOUR_USERID?totSecretKey=secret_YOUR_SECRET_KEY&totApiKey=public_YOUR_PUBLIC_KEY&appDomain=your.domain.com&scope=seal-details"

--- Response ----

{
   "content": {
       "reasons": {
           "isUserBaselineVerified": {
               "value": true,
               "dependsOn": [
                   "isOnlineVerified"
               ]
           },
```

```
        "isRealWorldVerified": {
            "value": false
        },
        "isUserBaselineVerified": {
            "value": true,
            "dependsOn": [
                "accountDataIsConsistent",
                "isOnlineVerified"
            ]
        },
        "accountDataIsConsistent": {
            "value": true,
            "dependsOn": [
                "familyNameIsConsistent",
                "givenNameIsConsistent"
            ]
        },
       "isOnlineVerified": {
        "dependsOn": [
          "accountDataIsConsistent",
          "facebook_totVerifiedAccount",
          "paypal_totVerifiedAccount"
        ],
        "value": "noMatch"
       },
       "facebook_totVerifiedAccount": {
        "dependsOn": [
          "facebook_isLongStanding",
          "facebook_accountDataIsConsistent",
          "facebook_hasHighQualityRealWorldLink",
          "facebook_hasEnoughConnections"
        ],
        "value": "noMatch"
       }
    },
    "consistency": {
        "baseline": "community",
        "compares": {
            "fullName": {
                "status": "match",
                "considered": {
                    "online": 2,
                    "primary": 1
                },
                "matches": {
                    "online": 2,
                    "primary": 1
                }
            },
            "givenName": {
```

```
                    "status": "match",
                    "considered": {
                        "online": 2,
                        "primary": 1
                    },
                    "matches": {
                        "online": 2,
                        "primary": 1
                    }
                },
                "familyName": {
                    "status": "match",
                    "considered": {
                        "online": 2,
                        "primary": 1
                    },
                    "matches": {
                        "online": 2,
                        "primary": 1
                    }
                }
            },
            "considered": {
                "online": 2,
                "primary": 1
            }
        }
    },
    "metadata": {
        "version": "1.0",
        "status": 200
    }
}
```


POST /api/connection/{appUserid}

Allows a community to create or update a connection to an end-user and set the metadata associated with that user.

The metadata is used to determine the match status of the community user with the TOT user during /users/{appId}/reputation requests. For details on how these attributes are used in reputation reports see the Matching section above.

The response will include an accessToken is returned which can be used to render the Account Connector Widget.

Request Spec

```
'spec': {
      method: ‘POST’,
      path: '/api/connection/:appUserid’,
      description: 'Allow the community to create a connection an end-user.',
      produces: ['application/json'],
      parameters: [
            param.path('appUserid', 'community ID of user', 'string'),
     ,
            param.body('appDomain', 'The domain registered for this api key.',
'string'),
            param.body('totApiKey', 'A valid apiKey.', 'string'),
            param.body('totSecretKey', 'A valid secret Key.', 'string'),
            param.body('appData', 'Community data for user.', 'AppData')]
}
```

AppData

The appData body has the following fields, but may include others.

```
appData : {
      location: LocationObject (see below),
      givenName: string or map of :’name’, or ‘nickname’ values (see Specifying
Names),
      familyName: string or map of: ‘maternal’, ‘paternal’, ‘maiden’ values
(see Specifying Names),
      middleName: string,
      dateOfBirth: {year: YYYY, month: MM, date: DD},     // FUTURE
      emailAddress: [/* array of up to 3 addresses */],   // FUTURE
      phoneNumber: [/* array of up to 3 phone numbers */] // FUTURE
}
```

NOTE: We accept but don’t yet match against dateOfBirth, emailAddress and phoneNumber. These fields should be provided if they’re available so that when consistency and reasoning for these fields are implemented that these values are already available.

LocationObject

```
            location : {
                  locality: “Bogota”,
                  region: “Columbia”,
                  countryCode: “CO”,
                  postalCode: “”,
                  line1: “”,
```

```
        line2: ""
}
```

Specifying Names

In the simplest case a names are represented as a string this way:

```
familyName: "Smith",
givenName: "John"
```

Unfortunately a significant part of the population cannot describe their name identity quite so simply. Last names (aka family names) are handled differently throughout the world. Family names change differently depending upon culture, marriage and even legal name changes. Beyond this historically people protected themselves using aliases when interacting with strangers online and this practice is still common. Lastly, many people have nicknames that they use in particular on social networks and when talking with close friends.

In order for name compares to be useful and not too restrictive Token of Trust has to behave well when confronted with these variations. To handle these cases we support an object base name-map to allows the integrator to specify different names for different reasons. When this is done we are able to match different variations of names without requiring user intervention.

For ‘familyName’ we support these keys:

current - the current last name - this is the assumed key if a the family name is a string. This is the key used to determine a full match with the user’s family name - matching against other attributes will result in a partial match and potentially a reason to be careful.
paternal - last name from father’s side.
maternal - last name from mother’s side.
maiden - name prior to marriage
previous - a previous last name
alias - one or more alternate names (can be a string or array of strings).

For ‘givenName’ we support these keys:

current - the current given name - this is the assumed key if a the family name is a string. This is the key used to determine a full match with the user’s family name - matching against other attributes will result in a partial match and potentially a reason to be careful.
previous - a previous given name
nickname - one or more nicknames (can be a string or array of strings).
alias - one or more alternate names (can be a string or array of strings).

Paternal/Maternal Matching

In Latin America for example we might expect many people to have 2 family names - one from the paternal side and one from the maternal side. This should be passed to Token of Trust this way:

```
familyName : {
      paternal: “Gonzales”,
      maternal: “Hernandez”
}
```

When Token of Trust compares a name from a social network to a paternal/maternal familyName the default order of the comparison will be paternal maternal, e.g. in the above example “Gonzales Hernandez”.

Here’s how Token of Trust would match against incoming names (from Social Networks for example):

```
“Gonzales Hernandez” : fullMatch.
“Gonzales” : partialMatch.
“Hernandez” : partialMatch.
“Hernandez Gonzales” : noMatch.
```

In the event that these names should be considered in opposite order the ‘current’ name should be specified to override the default ordering of paternal, maternal - like so:

```
familyName : {
      paternal: “Gonzales”,
      maternal: “Hernandez”,
      current: “Hernandez Gonzales”
}
```

With this adjustment here’s how Token of Trust would match against the incoming names from the previous example:

```
“Gonzales Hernandez” : noMatch.
“Gonzales” : partialMatch.
“Hernandez” : partialMatch.
“Hernandez Gonzales” : fullMatch.
```

In short, to get a fullMatch the complete name must be specified in the correct order - otherwise a partialMatch is returned.

Maiden Names

In the United States we typically think of having just one last name but it is very common to have a maiden name, a previous last name or both. These can be specified this way:

```
familyName : {
      current: “Smith-Johnson”,
      previous: “Jameson”,
      maiden: “Johnson”
}
```

As mentioned - a match against previous or maiden is considered a partial match and the system may reveal in a public profile that you’re using a past or maiden name and raise a reason to be careful.

Hyphenated Names

Sometimes people choose to take on hyphenated names. This is common practice in the United States after marriage. These can be specified this way:

```
familyName : {
      current: “Smith-Johnson”
}
```

Here’s how Token of Trust would match against incoming names (from Social Networks for example):

```
“Smith-Johnson” : fullMatch.
“Smith” : partialMatch.
“Johnson” : partialMatch.
“Johnson-Smith” : noMatch.
```

If the integrator has knowledge of maiden name it should be specified because in the user’s Reputation Report it can then be called out as a more credible reason for a less than full match. In this case that would just be added as follows:

```
familyName : {
      current: “Smith-Johnson”,
      maiden: “Johnson”
}
```

The matching itself doesn’t change:

```
“Smith-Johnson” : fullMatch.
“Smith” : partialMatch.
```

```
"Johnson" : partialMatch.
"Johnson-Smith" : noMatch.
```

Aliases

In our early user feedback sessions we learned not trusting social networks, institutions and strangers to be responsible about protecting their data many people use aliases to protect themselves.

We allow that as follows:

```
familyName : {
      current: "Brown",
      alias: "Braun"
}
```

- Or -

```
familyName : {
      current: "Brown",
      alais: ["Braun", "Browne"]
}
```

Matching against this results in the following:

```
"Brown" : fullMatch.
"Braun" : partialMatch.
```

Use of 'Braun' in the case above is also likely to result in a reason to be careful. Also note that 'Browne' would be a 'noMatch' in the first configuration and a 'partialMatch' in the second.

Nicknames

In addition to 'alias', givenName also allow the 'nickname' field which can either be a string or an array:

```
givenName : {
      current: "Johnathan",
      nickname: "Johnny"
}
```

- Or -

```
givenName : {
      current: "Johnathan",
      nickname: ["J-Dog", "Jon", "Jonny", "Johnny"]
}
```

Note that the use of multiple names may result in the appearance of one or more reasons to be careful. The difference between nickname and alias is only in how we explain the use of them in reasons to be careful… e.g. ‘this user is using a nickname on Facebook’ vs ‘this user has chosen not to use their real name on Facebook’... nicknames are generally considered more trustworthy than aliases but both will be considered partialName matches.

The Use of Aliases and Nicknames

As of this writing we are considering how to balance the use of aliases and nicknames. In both cases we are considering sharing their values in profiles and APIs to ensure that we strike the right balance of transparency, flexibility and prevent abuse. For example - we are considering listing one or more reasons to be careful if people use nicknames or aliases but lifting this reason if they allow the full set of nicknames and aliases to be shared with the viewer.

Examples of Name Matching and Results

Here are some concrete examples to keep in mind as you think about how name matching works:

1. Names compares are case insensitive (Smith === smith).
2. Names compares ignore special characters. (O'Brien === OBrien).
3. Names compares ignore diacritics. (Jesús === Jesus).
4. The "current" field is always used by default: {current : "Blanco Hernández"} matches "Blanco Hernández"
5. If no "current" field we assume "paternal maternal" : {paternal: "Blanco", maternal: "Hernández"} matches "Blanco Hernández"
6. When we use "paternal maternal" they must be in that order: {paternal: "Blanco", maternal: "Hernández"} does NOT match "Hernández Blanco".
7. If paternal-maternal attributes are out of order specify a "current" : {current: "Hernández Blanco", paternal: "Blanco", maternal: "Hernández"} does match "Hernández 8. Blanco".
8. "current" always takes precedence over paternal-maternal attributes: {current: "Hernández Blanco", paternal: "Blanco", maternal: "Hernández"} does NOT match "Blanco Hernández"
9. If we find only one of paternal or maternal it will partially match: {paternal: "Blanco", maternal: "Hernández"} partially matches "Blanco".
10. If we find only one part of a two part "current" name it will partially match: {current : "Blanco Hernández"} partially matches "Blanco".
11. If we find only maternal it will partially match: {paternal: "Blanco", maternal: "Hernández"} partially matches "Hernández".
12. If we find only one part of a two part "current" name it will partially match: "Hernández" partially matches {current : "Blanco Hernández"}.

13. Two part "current" names must always be in order: "Hernández Blanco" does not match {current : "Blanco Hernández"}.
14. > Two part "current" names must be exact matches: "Hernández Blanco" does not match {current : "Blanco Hernández"}.
15. substrings should not partially match (e.g. Smithe !== Smith).
16. Smith-Kline fully matches baseline Smith-Kline
17. Smith partially matches baseline Smith-Kline
18. Kline partially matches baseline Smith-Kline
19. Smith-Kline partially matches baseline Smith.
20. Smith-Kline partially matches baseline Kline.
21. compound Smith-Kline does NOT partially match Smithe-Kline.
22. test Smith-Kline does NOT partially match Kline-Smith.
23. alternate names partially match when exact matches are not present.

Response

If successful the response includes an accessToken is returned which can be used to render the Account Connector Widget. The following will be returned at minimum in the 'content' section of a successful response:

```
"accessToken":"ACCESS_TOKEN",
"expires":"EXPIRATION_TIMESTAMP"
```

Example Curl Request for POST /api/connection

```
darrin ~/src/tot/tot$  curl -X POST -H "Content-Type: application/json" \
            -d '{
    "appData" : {
        "givenName": "Darrin",
        "location" : {
            "locality" : "Minneapolis",
            "countryCode": "US"
        }
    }
}'
"https://app.tokenoftrust.com/api/connection/YOUR_USERID?totSecretKey=secret_YOUR_SECRET_KEY&totApiKey=public_YOUR_PUBLIC_KEY&appDomain=YOUR_DOMAIN"

--- Response ---

{
```

```
   "content": {
        "createTimestamp": 1465067807895,
        "expiresTimestamp": 1465235156169,
        "updatedTimestamp": 1465148756169,
        "accessToken":
"eyJlbmNyeXB0ZWQiOnsiZW5jcnlwdGVkRGF0YSI6IjFVTU9Zc1NTL3U0dGdLUTJJRWllWW53OUx4S0ZlWlFTa
G5OaUtDb29RbW9id3JKTGozNzZhTnBuQzZoajY5WXFhbkgxVUszUGFTUUZGVHpCdFZTSHdIbUpja0d3dWJIT29
NUXNKeWFpYWxQdTdqVzNZN2hnUU4rYUZwb3dIMG9Hb1FBREYxOW5FbHg4OHVJY1VRVmJ3VEhtOGdNNlYvU1hsO
FRqY3Rpb25Nd0dFNnpwbms3ZThWaUJTYkU0RVh1VitLb3hHa2ZzTERVVHFNQ2FRQm1Od1FqM00vUWhtRHNIUEF
jcENaWmdqcXExZkFrWUlJbmFVQ2lwM00rczNUdjRHTDhZZVdFaFgxRDRpa1I1M2ttdFFYbzd1UVIwRUFoL1Awb
1R3OFJ0RHlkSnRSaVhSMmxobi9ZYmordGpocVBJcjkvZGZ5WUNzcDQ3ZVZJTkYxVTltZXZIVkFvVTdDNFliZkR
DR0NvSitpU3dxNnVUOTY3ODhJTVgrWmovN01qL0tPV3A0NjZCR29reHdYK2ZqVVc0QVQxNTR0V1B5V0drbHkvb
kplZWRoZmtmRnZ2a3JQVnBxVzdSNFdWT2d6YUJkbHRZUkNzV0FEWldvcEFKL1lzdFNocVRtWW5Ec09ZOGJ1ZTd
2amNPellISXNkbHFzQmFRSmowTzlId2RGdXNoIiwicGFzc3dvcmRTYWx0IjoicmxNWGcxN1k5RTg5MkFrUU5wR
EczSjVpdytYck16WE43b3cvam40TC8xYz0iLCJjaXBoZXJJdiI6InlHRHNMcFZZeWxaUm5kSmxta09BekE9PSI
sImhtYWMiOiJWb0J4NGVpN3h3RnV1UUg4N1hOTm5UY2Z6dGdsM21yNDFwNXhkZFcrVkNnPSJ9LCJjcmVhdGVkI
joxNDY1MTQ4NzU2MjQ4LCJhcGlLZXkiOiJwdWJsaWNfdGVzdC1adXJtYXRTdXNweU9hcFlTRFYiLCJyZWZlcnJ
lck9yaWdpbiI6InRlc3QuZmluZGFwbGFjZS54eXoifQ=="
   },
   "metadata": {
        "version": "1.0",
        "status": 200
   }
}
```


GET /api/connection/{appUserid}

Allows a community to fetch the user data associated with a connection to an end-user - if one was previously set with /api/connection.

Request Spec

```
'spec': {
      method: 'GET',
      path: '/api/connection/:appUserid',
      description: 'Allow the community to get connection information related
to an end-user.',
      produces: ['application/json'],
      parameters: [
            param.path('appUserid', 'community ID of user', 'string'),
,
            param.body('appDomain', 'The domain registered for this api key.',
'string'),
            param.body('totApiKey', 'A valid apiKey.', 'string'),
            param.body('totSecretKey', 'A valid secret Key.', 'string')]
```

```
}
```

Response

If successful the response includes an accessToken is returned which can be used to render the Account Connector Widget. The following will be returned at minimum in the 'content' section of a successful response:

```
"appData":{ /** see AppData **/ },
"createTimestamp": 1465067807895,
"expiresTimestamp": 1465312366727,
"updatedTimestamp": 1465225966727,
```

Example Curl Request for GET /api/connection

```
curl -X GET -H "Content-Type: application/json"
"https://app.tokenoftrust.com/api/connection/demo-darrin?totSecretKey=secret_YOUR_SECRET_KEY&totApiKey=public_YOUR_PUBLIC_KEY&appDomain=YOUR_DOMAIN"

--- Response ---

{
  "content": {
    "createTimestamp": 1465067807895,
    "expiresTimestamp": 1465312366727,
    "updatedTimestamp": 1465225966727,
    "appData": {
      "givenName": "Darrin",
      "familyName": "Edelman",
      "location": {
        "countryCode": "US",
        "locality": "Minneapolis",
        "regionCode": "MN"
      }
    }
  },
  "metadata": {
    "version": "1.0.1",
    "status": 200
```

```
    }
}
```

References

We use YARAS as a basis for our RESTful APIs.

Reasons Descriptions

Below is a description of the most common descriptions as of today. The mathematical formula for each of these may deviate slightly from these english language descriptions.

isUserBaselineVerified - the user is at least online verified (isOnlineVerified) - when this is true the verified indicator widget will show up and display that the user is verified and the reputation summary will show up and display the users status (i.e. online and real world verification status).

isOnlineVerified - is true if we have at least two verified online accounts and when the data provided to the account via `/api/connection` is consistent with the information we have on file. Applicable online accounts include facebook, paypal and googlePlus. Alternatively the user may have one verified account and real world verification with driver's license or passport. Note that if this user is on OFAC terrorist watchlist this value is automatically false.

When false, the Token of Trust profile and the widget's on the community will appear in their 'Not Verified' state. When true Token of Trust widgets will appear. Specifically the verified indicator widget will show up and display that the user is verified and the reputation summary will show up and display the user's status.

isRealWorldVerified - the user has verified a real world driver's license, passport or other official identity (e.g. voter ID card in Mexico) and the data provided to the account via `/api/connection` is consistent with the information we have on file.. Note that if this user is on OFAC terrorist watchlist this value is automatically false.

accountDataIsConsistent - when true the community data is consistent with the information Token of Trust has for this user.

givenNameIsConsistent - when true the community 'givenName' is consistent with the givenName Token of Trust has for this user.

locationIsConsistent - when true the community location is consistent with the location Token of Trust has for this user. Please note: this considers the primary locations for each social network not social network 'activities'.

facebook_totVerifiedAccount - when true the user has shown a credible long standing presence on facebook. Criteria for this include: 1 or more year of account history, a minimum of 15 friends, a login to prove ownership within the last year, as well as consistent age, name, gender and primary location - if these are supplied within the baseline.

googlePlus_totVerifiedAccount - when true the user has shown a credible long standing presence on googlePlus. Criteria for this include: 1 or more year of account history, a login to prove ownership within the last year, as well as consistent age, name, gender and primary location - if these are supplied within the baseline.

paypal_totVerifiedAccount - when true the user has shown a credible long standing presence on paypal. Criteria for this include: 1 or more year of account history, a login to prove ownership within the last year, as well as consistent age, name, gender and primary location - if these are supplied within the baseline.

facebook_

Internal Reference:

https://docs.google.com/document/d/12IzR0Qk4mZqJTKVFOCyROnlsm_-15UV3UgR36y3Dp3E/edit#heading=h.fm3x3p4vfutp