Nimble API Documentation

Release 1.3.0

Nimble

Mar 23, 2018

Contents

Actual Nimble API version - *1.3*

Contents:

CHAPTER 1

Overview

Nimble is a social relationship management tool that brings together your contacts from many disparate locations into one place; gives you a dynamic, 360º view of each contact; and helps you nurture the new and important contacts in your network. The Nimble platform offers access to the stored data via our APIs. Currently, only contact data is accessible via our REST API, and this data is available with limited functionality under the "beta" label. We plan on releasing additional functionality in the following order:

- **GET contact list and contact detail**, includes simple SEARCH – available
- **CREATE/UPDATE/DELETE contact** - available
- **GET contact list** with advanced SEARCH - available
- **GET/CREATE/UPDATE/DELETE custom fields and groups** via metadata API - available
- **GET/CREATE/UPDATE/DELETE notes related to contacts** - available
- **CREATE related activities** - available
- **GET related activities** - TBD
- **GET related messages** - TBD
- **GET related social** - TBD
- **MERGE contact** - TBD

Given the early state of the API, some aspects of the API are subject to change.

We are building the API with a few main use cases in mind: widgets/extensions to Nimble, web clients (including our own), mobile clients. and ultimately 2-way data integrations with other services. We aspire to make an API that is simple to use, easy to read, and flexible.

Your feedback is greatly appreciated while we continue to shape our API offering.

CHAPTER 2

Nimble API Responses

Contents

2.1 Contacts details

Typical response to this request is a dictionary with 2 keys (unless otherwise specified by the specific API): meta and resources.

2.1.1 Contact resources

This field usually contains all data for the contacts you've requested. Here is an example of a Nimble contact

```
'resources': [{
    'updated': '2012-09-07T16:49:56+0300',
    'created': '2012-09-07T16:49:56+0300',
    'fields': {
```

```
        "parent company": [
            {
                "group": "Basic Info",
                "name": "parent company",
                "label": "parent company",
                "modifier": "",
                "presentation": {},
                "id": "5049f697a694620a0700004d",
                "multiples": false
            }
        ],
        'description': [{
            "group": "Extra Info",
            "name": "description",
            "label": "description",
            "modifier": "other",
            "presentation": {},
            "id": "5049f697a694620a0700007f",
            "multiples": true
        }, {
           "group": "Extra Info",
           "name": "description",
           "label": "description",
           "modifier": "linkedin",
           "presentation": {},
           "id": "5049f697a694620a07000082",
           "multiples": true
        }],
        'last name': [{
            'field_id': '5049f697a694620a07000045',
            'modifier': '',
            'group': 'Basic Info',
            'value': 'Akopyan',
            'label': 'last name'
        }],
        'phone': [{
            'field_id': '5049f697a694620a07000058',
            'modifier': 'mobile',
            'group': 'Contact Info',
            'value': '+7 (917) 202-456-1111',
            'label': 'phone'
        }, {
            'field_id': '5049f697a694620a07000056',
            'modifier': 'home',
            'group': 'Contact Info',
            'value': '+7 244 231 84 22',
            'label': 'phone'
        }],
        'URL': [{
            'field_id': '5049f697a694620a0700007d',
            'modifier': 'other',
            'group': 'Extra Info',
            'value': 'https://nimble.com',
            'label': 'URL'
        }, {
            'field_id': '5049f697a694620a0700007d',
            'modifier': 'other',
            'group': 'Extra Info',
            'value': 'https://app.nimble.com',
            'label': 'URL'
        }],
        'source': [{
            'field_id': '5049f697a694620a0700004f',
```

```
            'modifier': '',
            'group': 'Basic Info',
            'value': 'csv',
            'label': 'source'
        }],
        'address': [{
            'field_id': '5049f697a694620a07000075',
            'modifier': 'other',
            'group': 'Contact Info',
            'value': '{"city": "Dushanbe", "street": "First str. 15", "zip": "54055
↪", "country": "Farganistan"}',
            'label': 'address'
        }],
        'email': [{
            'field_id': '5049f697a694620a07000065',
            'modifier': 'other',
            'group': 'Contact Info',
            'value': 'fake_person@nimble.com',
            'label': 'email'
        }],
        'first name': [{
            'field_id': '5049f697a694620a07000043',
            'modifier': '',
            'group': 'Basic Info',
            'value': 'Amayak',
            'label': 'first name'
        }]
    },
    'object_type': 'contact',
    'id': '5049fb849b85f669e40000dc',
    'last_contacted': {
        'last_contacted': '2012-09-17T11:43:51+0300',,
        'thread_id': 5049f697a694620a07000062,
        'message_id': 5049f697a694620a17000075
    },
    'avatar_url': 'https://app.nimble.com/api/contacts/avatars/
↪5049fb849b85f669e40000dc'
    'record_type': 'person',
    'creator': 'Emil Kio',
    'children': [],
    'tags': [{
        'tag': 'csv import',
        'id': '5049fa0c9b85f62cb4000639'
    }],
    'owner_id': '5049f696a694620a0700001c'
}]
```

Here is a description of the response in detail:

updated Timestamp of contact's last update time

created Timestamp of contact's creation time

fields Dictionary containing contact's fields data. Keys are field names and values are lists of field values. All default contact fields are *described here*.

object_type String specifying document type. For contacts it's `contact`.

id Unique contact id in BSON format.

last_contacted

Information about last outbound message to this contact (if any). Contains following fields.

- *last_contacted* — timestamp of last outbound message

- *thread_id* — unique id of message thread in BSON format
- *message_id* — unique id of message in BSON format

avatar_url URL of image that can be used as contact's avatar. Value of null is used to indicate that contact has no avatar associated.

record_type Type of contact. This can have one of two values: `person` and `company`.

creator Name of the person who created the contact

children For `company` contacts this field contains list of `person` contacts associated with the company.

tags

List of tags associated with the contact. Each tag is represented as a dictionary having following keys.

- *tag* — tag's text
- *id* — unique id of tag in BSON format

owner_id Id of the person owning the contact in BSON format

2.1.2 Contacts metadata

Contact's metadata contains information about all basic and custom fields created in Nimble for a user. Below is it's typical structure. Please note that this listing doesn't contain all metadata as the full list is very big. The typical records are shown here. All default contact fields are *described here*.

```
'contacts_meta': {
    'fields': {
        'first name': [{
            "group": "Basic Info",
            "name": "first name",
            "label": "first name",
            "modifier": "",
            "presentation": {},
            "id": "5049f697a694620a07000043",
            "multiples": false
        }],
            'email': [{
            "group": "Contact Info",
            "name": "email",
            "label": "email",
            "modifier": "other",
            "presentation": {},
            "id": "5049f697a694620a07000065",
            "multiples": true
        }, {
            "group": "Contact Info",
            "name": "email",
            "label": "email",
            "modifier": "personal",
            "presentation": {},
            "id": "5049f697a694620a07000064",
            "multiples": true
        }],
        'lead status': [{
            "group": "Lead Details",
            "name": "lead status",
            "label": "lead status",
            "modifier": "",
            "presentation": {
                "width": "1",
```

```
                "next_id": "5",
                "values": [
                    {
                        "id": "1",
                        "value": "Open"
                    },
                    {
                        "id": "2",
                        "value": "Contacted"
                    },
                    {
                        "id": "3",
                        "value": "Qualified"
                    },
                    {
                        "id": "4",
                        "value": "Unqualified"
                    }
                ],
                "type": "select-box"
            },
            "id": "5049f697a694620a0700008d",
            "multiples": false
        }]
    },
    'groups': {
        'Basic Info': {
            "name": "Basic Info",
            "order": [
                "first name",
                "last name",
                "middle name",
                "company name",
                "title",
                "parent company",
                "source",
                "last contacted"
            ],
            "is_standard": true,
            "label": "Basic Info",
            "type": "both",
            "id": "5049f696a694620a07000031"
        }
    }
}
```


Here is a description of the response in detail:

fields Information about the fields in Nimble. Represented by dictionary where keys are fields names, and values are lists containing details about all possible modifications of this field. If field have no modifiers (like `first name` on example above), this list contains only one element.

Information stored in dictionaries with following keys:

- *group* — unique name of the group containing this field.
- *label* — unique name representing the field in human-readable form.
- *modifier* — name of the field's modifier
- *id* — unique id of the field in BSON format
- *multiples* - indicates whether field could have multiple values (under different modifiers).
- *presentation* - dict with the information which should help to display this field on client. More

details are at *described here*.

groups

Information about field groups. Represented by dictionary where keys are unique group names and values are diction

- *id* — unique id of the group in BSON format.
- *order* — list containing names of the fields as they appeared in group.
- *name* — unique name of the group. (Outdated: as we have field name as the key of `groups` dictionary.)
- *label* — unique name representing the field in human-readable form.
- *is_standard* - whether this group belongs to standard Nimble groups.
- *type* - type (belonging) of group, could be among `person`, `company`, `both`.

2.2 Contact list

Contact list request is similar to *Contacts details*. It has the same key with resources, *described here*. Difference is in `meta` key value. For contact listing it returns pagination details.

2.2.1 Pagination metadata

```
'meta': {
    'per_page': 30,
    'total': 45,
    'pages': 2,
    'page': 1
}
```

Keys meaning:

per_page Number of contacts returned per page

total Total number of contacts

pages Total pages count

page Current page number

2.3 API Errors

Errors in Nimble are returned as a JSON dictionary with appropriate HTTP error codes and following keys:

message Message about the error

code Extended error code

2.3.1 Validation Error

Sent on invalid parameters. Returns with HTTP code 409 and code field equal to 245.

This response looks like common error dictionary:

```
{
    "message": "You can specify either `keyword` or `query` parameter, not both!",
    "code": 245
}
```

On contact creation and update — additional data is returned.

```
{
    "message": "Validation errors",
    "code": 245,
    "errors": {
        "first name": [{
            "message": "First name or last name field is required for person and␣
↪should not be empty",
            "field_id": "5049f697a694620a07000043"
        }]
    }
}
```

Here errors are a dictionary, containing information about field that caused the error. Key is field name and values are extended error message and unique id of the field that caused the error.

2.3.2 Quota Error

Sent if user exceeded his quota values. Returns with HTTP code 402 and code field equal to 108.

```
{
    "message": "You have created the maximum number of contact records allowed for␣
↪your subscription.\nDon't worry, you can upgrade your account and add more␣
↪contacts right now.",
    "code": 108
}
```

2.3.3 Server error

Sent if unrecoverable Nimble server occurs. Returns with HTTP code 500 and code field equal to 107.

```
{
    "message": "Internal error handling request",
    "code": 107
}
```

2.3.4 NotFound Error

Sent on attempt to get some object by invalid identifier (in most cases identifier of object is its ID in our database).

This response will contain dictionary with *object_type* and *object_id* fields:

```
{
    "object_type": "contact field",
    "object_id": "111111111111111111111111"
}
```

CHAPTER 3

Get user information

For many use-cases it is useful to obtain basic user information. For that purpose we provide the *GET /api/v1/myself* endpoint.

3.1 Request

Example:

```
GET https://api.nimble.com/api/v1/myself
```

3.2 Response: OK

On success, server returns response with HTTP code 200

```
{
  "user_id": "4f2acc3142a053dda595f00b",
  "company_id": "4c2118ad54397f271b000000",
  "email": "some@user.com",
  "name": "John Doe",
  "company_size": 3
}
```

CHAPTER 4

Contacts API

Contents:

4.1 Basic Contacts API

Contents:

4.1.1 Get contacts list

Request

For contacts listing we support two endpoints: base, returning full contact info, and ids-only endpoint that return only contact ids. Last one works faster then base one, so if you need only ids — please use it.

Base endpoint:

```
GET https://api.nimble.com/api/v1/contacts/
```

IDs endpoint:

```
GET https://api.nimble.com/api/v1/contacts/ids/
```

Parameters

All parameters are optional. Unrecognized parameters are ignored. Unrecognized values will return an error.

fields — default: all fields in contact

> Specifies a comma separated list of fields to return. If this parameter is excluded, all fields will be returned. For example: `fields=first%20name,my%20custom%20field`. For more detailed info on Nimble's fields see *Nimble default fields*.
>
> ---
>
> **Note:** If field name contains "," (comma) it should be shielded with "". For example: we have some custom field with name "hello, Jon Doe" it should be HTML-encoded in

`hello%5C%2C%20John%20Doe` (`hello\, John Doe`).

tags — default: 1

Specifies whether tags should be included in the results.

per_page — default: 30

Specifies the number of items to return per page of results.

page — default: 1

Specifies which page to display. Numeration starts from 1.

sort — default: none

Identifies the sort field and sort order. Sort order is required when this parameter is used. An single sort field can be specified. Any field can be sorted in either `asc` or `desc` order. All *searchable fields* which aren't multiple and aren't custom fields are sortable.

Note: There is no way to sort requests which have size more then 100 contacts. In order to sort results you should set `per_page` with number less or equal 100.

We suggest to use `score` sorting if `contain` type of occurrence or `keyword` parameter is used.

Information about if field is multiple can be retrieved from fields metadata. There are some notes for special fields:

Table 4.1: Special sortings

Name	Meaning	Note
recently viewed	sorts by user's recently viewed contacts	When sorting by recently viewed, these parameters are disabled: **keyword**, **record type**, **page** and **per_page**. Nimble stores only the 30 most recently viewed records.
score	sorts by relevance to search request	This sorting only have sense when you are performing search request with *contains* type of occurrence.
users last contacted	sorts by recently contacted by user contacts	When sorting by user's last contacted, these parameters are disabled: **keyword** and **record type**.

record_type — default: `all`

Identifies the record type. Possible value are `person`, `company`, and `all`.

keyword — default: empty

Specifies a set of simple search criteria for the query. This simple search is performed on any (indexed in our search engine) fied of contact. For example: `keyword=Jon%20Smith` We suggest to use `score` sorting with this parameter.

query — default: empty Specifies query for contacts advanced search. Please note, that this parameter not compatible with parameters `record_type` and `keyword`. For more details about search see *Search contacts*.

Response: OK

List and Detail response format is the basically the same. List allows search terms, sort orders, and fields as parameters, whereas detail returns all of the fields with the option of adding metadata. In more details, this format *described here*.

Example response for IDs only request:

```
{
    "meta": {
        "page": 1,
        "pages": 1,
        "per_page": 30,
        "total": 2
    },
    "resources": [
        "4f69fb852ab3740c5e000004",
        "5e69fb852ab3f40d5e050017"
    ]
}
```

Response: Errors

Possible errors:

- *Validation Error*

4.1.2 Search contacts

Basic concepts

All values putted to search (values of the objects by which search is performed and the text of search request) are converted to lowercase and are subjected to procedure of ascii folding

Examples:

1. "cAr" is the same as "car"
2. "čar" is the same as "car"
3. "ČAR" is the same as "car"

No any another normalization procedures are used. (plural, phonetic, etc)

For some fields like social network profiles, emails, phones special chars escaping are being done.

Advanced search query syntax

Query language for Advanced search is JSON encoded structure.

Short example of querying all persons with skype "john.doe":

```
{
   "and": [
        {
            "skype id": {
                "is": "john.doe",
             }
        },
        {
            "record type": {
                "is": "person"
            }
        }
   ]
}
```

For more examples, see *examples*.

Terminology

Based on example above, let's define basic terminology:

and is join operator for occurrences

skype id (`john.doe`) is a term to search by

is is occurrence of term in search index

record_type (`person`) is a value for occurrence

Joins

Possible variants for join operators are `and` and `or`. They could be combined in different ways and priorities. Some examples with explanations will be listed below.

Let's define several occurrences:

```
o1 = {
    "skype id": {
        "is": "john.doe"
    }
};

o2 = {
        "record type": {
            "is": "person"
        }
},

o3 = {
    "first name": {
        "contain": "John"
    }
},

o4 = {
    "created": {
        "range": {
            "start_date": "2012-02-13",
            "end_date": "2012-02-23",
        }
    }
}
```

Join like o1 and o2 and o3 and o4:

```
{
   "and": [o1, o2, o3, o4]
}
```

Join like (o1 and o2) or o3 and o4:

```
{
   "and": [o4, {"or": [o3, {"and": [ o1, o2 ] } ] } ]
}
```

Join like (o1 and o2) or (o3 and o4):

```
{
   "or": [{"and": [ o1, o2 ] }, {"and": [ o3, o4 ] }]
}
```

Note: Maximum limit of occurrences in one request query is 11; If request could be done without join operators — then it should contain only single occurrence.

Search operators

Note: To get relevant results you may use *sorting* by relevance.

Note: In the table below `(old)` means only another(previously used) behaviour of search operator, API parameters is still same.

Table 4.2: Full list of available search operators

Operator	Description	Example
contain	looks for EXACT match of ANY word from provided request in the words of specified contact field. Only WHOLE words from query and contact data are used. There is no additional analysis for part of word.	Some of words in provided search request (one or more) for specified field is equal to some word (one or more) in field of contact (contacts). This contacts will be returned as result of search request. For example you are searching for `Jon Pupken` in name field So contacts with the following name will be matched: `JON PUPKEN`, `JON travolta`, `James PUPKEN` Contacts with this names will not be matched: `JaN PUPKEr`, `JONy PoPov` As more equal words from request string are in contact field as higher contact is in returned list if *sorting* set to *by relevance in descending order*. Query exapmle: `{"address": {"contain": "Greater LA"}}`
contain(old)	Provided value matches field value from LEFT OR RIGHT side. For example `*document_value` or `document_value*`. But not both.	For example you are searching for `POPOV` in last_name field So contacts with the following name will be matched: `POPOV`, `POPOVa`, `POPOVenko`, `podPOPOV` Contacts with this names will not be matched: `PuPken`, `podPOPOVenko` Query example: `{"first name": {"contain": "aaa"}}`
starts_with	Provided value matches field value from LEFT side. For example `document_value*`.	For example you are searching for `POPOV` in last_name field So contacts with the following name will be matched: `POPOV`, `POPOVa`, `POPOVenko` Contacts with this names will not be matched: `PuPken`, `podPOPOVenko`, `podPOPOV` Query example: `{"first name": {"starts_with": "value"}}`
is	Provided value is equal to field value	`{"record type": {"is": "all"}}`
is_empty	Feild value with specified name is absent or empty	`{"last name": {"is_empty": True}}`
in_the_last	Date field value of matched documents is within last X days/weeks/monthes	`{"created": {"in_the_last": {"unit": "day", "quantity": 2}}}`
range	Date field value of matched documents is within specified period. There are two types of selector for range occurrence type. `date` - simple case. Provided date will be converted to user timezone. Expected format is `%Y-%m-%d` `datetime` - provided date is expected to be in UTC in rfc3339 format.	`{"company last contacted": {"range": {"start_date": "2013-03-19", "end_date": "2013-03-19"}}}` `{"company last contacted": {"range": {"start_datetime": "2013-04-23 00:00:10", "end_datetime": "2013-04-26T00:00:10"}}}`
gt	Field value with specified name have lower value than provided in the search criteria	`{"rating": {"gt": "3"}}`
lt	Field value with specified name have greater value than provided in the search criteria	`{"rating": {"gt": "3"}}`
gte	Field value with specified name have lower or equal value than provided in the search criteria	`{"rating": {"gte": "3"}}`
lte	Field value with specified name have greater or equal value than provided in the search criteria	`{"rating": {"lte": "3"}}`

Available search fields

Table 4.3: Full list of available field types for searching on them

Field name	Possible operators
email	`is`, `is_not`, `contain(old)`, `not_contain(old)`, `is_empty`
skype id	`is`, `is_not`, `contain(old)`, `not_contain(old)`, `is_empty`
twitter	`is`, `is_not`, `contain(old)`, `not_contain(old)`, `is_empty`
linkedin	`is`, `is_not`, `contain(old)`, `not_contain(old)`, `is_empty`
facebook	`is`, `is_not`, `contain(old)`, `not_contain(old)`, `is_empty`
phone	`is`, `is_not`, `contain(old)`, `not_contain(old)`, `is_empty`
last name	`is`, `is_not`, `contain(old)`, `not_contain(old)`, `is_empty`
street	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
city	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
state	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
zip	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
country	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
company name	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
title	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
name	`is`, `is_not`, `contain(old)`, `not_contain`
first name	`is`, `is_not`, `contain(old)`, `not_contain`
lead source	`is`, `is_not`, `is_empty`
lead type	`is`, `is_not`, `is_empty`
lead status	`is`, `is_not`, `is_empty`
rating	`is`, `is_not`, `is_empty`, `gt`, `lt`, `lte`, `gte`
created	`in_the_last`, `range`
updated	`in_the_last`, `range`
company last contacted	`in_the_last`, `range`
address	`contain`, `not_contain`, `is_empty`
tag	`is`
custom_fields	`is`, `is_not`, `contain`, `not_contain`, `is_empty`
record type	`is`
description	`contain`, `not_contain`, `is_empty`

More search examples

Search all contacts with specified type:

```
{"record type": {"is": "person"}}
```

Search contacts with name, containing “Gal” and tagged with specific tag:

```
{
    "and": [{
        "first name": {
            "contain": "Gal"
        }
    }, {
        "tag": {
            "is": "csv import2"
        }
    }]
}
```

Search for contacts without values in *city* field:

```
{"city": {"is_empty": False}}
```

Search for contacts, created in given date range:

```
{
    "created": {
        "range": {
            "start_date": "2012-10-16",
            "end_date": "2012-10-18"
        }
    }
}
```

Search for specific value in custom field:

```
{"custom_fields": {"custom field1": {"is": "value"}}}
```

Note: If your custom field is `select-box`, in search you should specify not it's value, but id of this value. For example, if you have field with following values:

```
"values": [
    {
        "id": "1",
        "value": "Open"
    },
    {
        "id": "2",
        "value": "Closed"
    }
]
```

You should use `2` as value, if you want to find contacts with field equal to `closed`. For example:

```
{"custom_fields": {"comminication state": {"is": "2"}}}
```

Validation

To validate join operators, occurrences and values we're using "Json Schema" standard. Current implementation of rules is built with json-schema Draft 3. Please, use this draft for better understanding of query language rules.

In Nimble we're using json-schema python library to validate user search queries.

Also, on github you can find the library from one of the json-schema authors json-schema-validator. It's fully implementing draft 3 spec, and can be used as reference library.

Top level validation schema

```
{
    "additionalProperties": false,
    "patternProperties": {
            "^(email|skype id|twitter|linkedin|facebook|phone|last␣
↪name|title|description|street|city|state|zip|country|lead type|company␣
↪name|custom_fields|name|first name|lead source|created|address|tag|or|and|record␣
↪type)$": {
                    "required": true,
                    "type": "object"
            }
    },
    "type": "object",
```

```
    "description": "top level (all fields) validation rule"
}
```

Joins validation schema

```
{
    "additionalProperties": false,
    "patternProperties": {
            "^(or|and)$": {
                    "minItems": 2,
                    "type": "array"
            }
    },
    "type": "object"
}
```

Schema for validation of default fields occurrences

```
{
    "patternProperties": {
            "^(email|skype id|twitter|linkedin|facebook|phone|last␣
↪name|street|city|state|zip|country|company name|title)$": {
                    "additionalProperties": false,
                    "patternProperties": {
                            "^(is|is_not|contain|not_contain|is_empty)$": {
                                    "minLength": 2,
                                    "required": true,
                                    "type": ["string", "boolean"]
                            }
                    },
                    "type": "object"
            }
    },
    "type": "object",
    "description": "/twitter/linkedin/facebook/phone/last name/street/city/state/
↪zip/country/company name/title validation rule"
}
```

Schema for validation of full name/first name fields

```
{
    "patternProperties": {
            "^(name|first name)$": {
                    "additionalProperties": false,
                    "patternProperties": {
                            "^(is|is_not|contain|not_contain)$": {
                                    "minLength": 2,
                                    "required": true,
                                    "type": "string"
                            }
                    },
                    "type": "object"
            }
    },
    "type": "object",
    "description": "name/first name validation rules. name == first name + last␣
↪name"
}
```

Schema for validation of lead source/lead type field

```
{
    "type": "object",
    "description": "lead source/lead type validation rules",
    "patternProperties": {
            "^(lead source|lead type)$": {
                    "additionalProperties": false,
                    "patternProperties": {
                            "^(is|is_not|is_empty)$": {
                                    "required": true,
                    "type": ["string", "boolean"]
                            }
                    },
                    "type": "object"
            }
    }
}
```

Schema for validation of created occurrences

```
{
    "type": "object",
    "description": "created validation rule",
    "properties": {
        "created": {
            "type": [
                {
                    "type": "object",
                    "description": "sub-schema for validation range type occurrence
↪",
                    "properties": {
                        "range": {
                            "additionalProperties": false,
                            "required": true,
                            "type": "object",
                            "properties": {
                                "start_date": {
                                    "required": true,
                                    "type": "string",
                                    "description": "start date in format YYYY-MM-DD
↪",
                                    "format": "date"
                                },
                                "end_date": {
                                    "required": true,
                                    "type": "string",
                                    "description": "end date in format YYYY-MM-DD",
                                    "format": "date"
                                }
                            }
                        }
                    }
                },
                {
                    "type": "object",
                    "description": "sub-schema for validation in the last type␣
↪occurrence",
                    "properties": {
                        "in_the_last": {
                            "additionalProperties": false,
                            "required": true,
                            "type": "object",
                            "properties": {
                                "quantity": {
```

```
                                    "required": true,
                                    "type": "integer",
                                    "description": "quantity of units, like 10␣
↪days, 2 months etc"
                                },
                                "unit": {
                                    "required": true,
                                    "type": "string",
                                    "description": "possible types of period",
                                    "enum": ["day", "month", "week"]
                                }
                            }
                        }
                    }
                ]
            }
    }
}
```

Schema for validation of address occurrences

```
{
    "type": "object",
    "description": "address validation rule",
    "properties": {
            "address": {
                    "additionalProperties": false,
                    "patternProperties": {
                            "^(contain|not_contain|is_empty| )$": {
                                    "minLength": 2,
                                    "required": true,
                                    "type": ["string", "boolean"]
                            }
                    },
                    "type": "object"
            }
    }
}
```

Schema for validation of tag occurrences

```
{
    "type": "object",
    "description": "tag validation rule",
    "properties": {
            "tag": {
                    "additionalProperties": false,
                    "type": "object",
                    "properties": {
                            "is": {
                                    "minLength": 2,
                                    "required": true,
                                    "type": "string"
                            }
                    }
            }
    }
}
```

Schema for validation of custom fields

```
{
    "type": "object",
    "description": "custom field validation rule",
    "properties": {
            "custom_fields": {
            "type": "object",
            "patternProperties": {
                "^.{1,150}$": {
                    "additionalProperties": false,
                    "required": true,
                    "type": "object",
                    "patternProperties": {
                        "^(is|is_not|contain|not_contain|is_empty)$": {
                            "required": true,
                            "type": ["string", "boolean"]
                        }
                    }
                }
            }
        }
    }
}
```

Schema for validation of record type

```
{
    "type": "object",
    "description": "record type validation rule",
    "properties": {
        "record type": {
            "additionalProperties": false,
            "type": "object",
            "properties": {
                "is": {
                    "minLength": 2,
                    "required": true,
                    "type": "string",
                    "enum": ["all", "person", "company"]
                }
            }
        }
    }
}
```

Schema for validation of description

```
{
    "type": "object",
    "description": "description validation rule",
    "properties": {
        "description": {
            "additionalProperties": false,
            "patternProperties": {
                "^is_empty|contain|not_contain$": {
                    "minLength": 2,
                    "required": true,
                    "type": ["string", "boolean"]
                }
            }
        }
    }
}
```

API endpoints

Advanced search requests should be done through statard contacts listing entry point:

```
GET /api/v1/contacts
```

Parameters are the same as for regular listing, except new one:

query

Should contain url-encoded JSON. Syntax of queries is *described above*.

Note: Parameter `record_type` will be ignored, if `query` parameter was specified. To filter persons/companies, please use corresponding sub query in query.

Note: Parameter `keyword` will be ignored, if `query` parameter was specified.

Request example 1:

```
https://api.nimble.com/api/v1/contacts?query=%7B%22first%20name%22%3A%20%7B%22is%22
↪%3A%20%22Anton%22%7D%7D&tags=0&per_page=5&fields=first%20name
```

Advanced search query in this request is:

```
{
    "first name": {
       "is": "Anton"
    }
}
```

Request example 2:

```
https://api.nimble.com/api/v1/contacts?query=%7B%22and%22%3A%20%5B%7B%22last%20name
↪%22%3A%20%7B%22is%22%3A%20%22Ferrara%22%7D%7D%2C%20%7B%22first%20name%22%3A%20%7B
↪%22is%22%3A%20%22Jon%22%7D%7D%5D%7D&tags=0&per_page=5&fields=last%20name,first
↪%20name
```

Advanced search query in this request is:

```
{
     "and": [
          {
               "last name": {
                    "is": "Ferrara"
               }
          },
          {
               "first name": {
                    "is": "Jon"
               }
          }
     ]
}
```

Response: OK

On success, results are returned in format, similar to contacts *listing response*.

Response: Errors

Possible errors:

- *Validation Error*

4.1.3 Get contacts details

Single and bulk requests ore formatted and returned in the same way. The response format for each field is the same as returned in a contacts listing response. Metadata can be included in the response.

Request

Base endpoint:

```
GET https://api.nimble.com/api/v1/contact/<contact_id>[,<contact_id>,<contact_id>,.
↪..]
```

Example:

```
GET https://api.nimble.com/api/v1/contact/5049fb9b9b85f669e4000447,
↪5049fb7d9b85f669e4000066,5049fba29b85f669e40004fb
```

Parameters

All parameters are optional. Unrecognized parameters are ignored. Unrecognized values will return an error.

meta — default: 0 When included and equal to 1, the meta parameter will add an additional component to the response which describes all fields, field types, and possible values available on the record. For further reference see *Contacts metadata*.

fields — default: all fields in contact Specifies a comma separated list of fields to return. If this parameter is excluded, all fields will be returned. For example: `fields=first%20name,my%20custom%20field`. For more detailed info on Nimble's fields see *Nimble default fields*.

Note: If field name contains "," (comma) it should be shielded with "". For example: we have some custom field with name "hello, Jon Doe" it should be HTML-encoded in `hello%5C%2C%20John%20Doe` (`hello\, John Doe`).

---

tags — default: 1 Specifies whether tags should be included in the results.

Response: OK

List and Detail response format is the basically the same. List allows search terms, sort orders, and fields as parameters, whereas detail can only limit fields to return and provide the option of adding metadata. In more details, this format *described here*.

Response: Errors

Possible errors:

- *Validation Error*

4.1.4 Create contact

Request

Example:

```
POST https://api.nimble.com/api/v1/contact
```

Parameters

All parameters are passed as JSON in request body.

record_type — required Specifies the type of contact to create - the record type. This parameter could be one of two values: `company` or `person`.

fields — required Describes a dictionary organized in the same structure as a contact listing response. In this structure, each key is field name. Values are lists of dictionaries, having two fields: value - actual value to store in contact field, modifier - field modifier to use, if field can have one. At a minimum, contacts require a name (first or last for a person, company name for a company).

tags — optional, default: None Comma separated list of tags to assign to contacts. If you need to create tags, containing comma sign — escape it with backslash. E.g. `our customers,best\,premium` will create tags `our customers` and `best,premium`.

> **Note:** Maximum 5 tags are allowed in this list during contact creation.

avatar_url — optional, default: None String, pointing to avatar, that should be assigned to the contact.

> **Note:** Nimble uses lazy loading mechanism for avatars, and didn't perform any checks for URL validness during `avatar_url` setting. If you'll pass invalid parameter here — no avatar will be displayed for contact.

Example:

```
{
    "fields": {
        "first name": [{
            "value": "Jack",
            "modifier": ""
        }],
        "last name": [{
            "value": "Daniels",
            "modifier": ""
        }],
        "phone": [{
            "modifier": "work",
            "value": "123123123"
        }, {
            "modifier": "work",
            "value": "2222"
        }]
    },
    "record_type": "person",
    "tags": "our customers,best"
}
```

Response: OK

On success, server returns response with HTTP code 201 and newly created contact resource.


```
{
   "avatar_url": null,
   "children": [],
   "company_last_contacted": {
       "in": null,
       "out": null
   },
   "created": "2013-10-22T12:26:31+0300",
   "creator": "NimbleAPItest",
   "fields": {
       "first name": [
           {
               "field_id": "4eabb0b64fb88d334c000ab6",
               "group": "Basic Info",
               "label": "first name",
               "modifier": "",
               "value": "Jack"
           }
       ],
       "last name": [
           {
               "field_id": "4eabb0b64fb88d334c000ab8",
               "group": "Basic Info",
               "label": "last name",
               "modifier": "",
               "value": "Daniels"
           }
       ],
       "phone": [
           {
               "field_id": "4eabb0b74fb88d334c000ac5",
               "group": "Contact Info",
               "label": "phone",
               "modifier": "work",
               "value": "123123123"
           },
           {
               "field_id": "4eabb0b74fb88d334c000ac5",
               "group": "Contact Info",
               "label": "phone",
               "modifier": "work",
               "value": "2222"
           }
       ],
       "source": [
           {
               "field_id": "4eabb0b74fb88d334c000ac2",
               "group": "Basic Info",
               "label": "source",
               "modifier": "",
               "value": "m"
           }
       ]
   },
   "id": "526644c7837d4e249372f091",
   "is_important": null,
   "last_contacted": {
       "message_id": null,
       "thread_id": null,
```

```
        "tstamp": null,
        "user_id": null
    },
    "object_type": "contact",
    "owner_id": "4decc6b662100441e200000b",
    "record_type": "person",
    "reminder": null,
    "social_connections": {
        "facebook": {},
        "linkedin": {},
        "twitter": {}
    },
    "tags": [
        {
            "id": "52664434837d4e249372f081",
            "tag": "our customers"
        },
        {
            "id": "52664434837d4e249372f083",
            "tag": "best"
        }
    ],
    "updated": "2013-10-22T12:26:31+0300",
    "updater": null
}
```


For more details see: *Contact resources*.

Response: Errors

Possible errors:

- *Validation Error*
- *Quota Error*

4.1.5 Update contact

Request

Example:

```
PUT https://api.nimble.com/api/v1/contact/<id>?replace=1
```

Parameters

replace Optional url parameter that identifies whether to replace all other values for this kind of field or not. Can take 1 or 0 as true or false state, default 0. For example if contact has such set of fields:


```
{
    "fields": {
        "first name": [{
            "value": "Jack",
            "modifier": ''
        }],
        "email": [{
            "value": "user@nimble.com",
            "modifier": "work"
        }, {
```

```
            "value": "jack@gmail.com",
            "modifier": "personal"
        }]
    }
}
```

then update it with:

```
{
    "fields": {
        "email": [{
            "value": "user@nimble.com",
            "modifier": "personal"
        }]
    }
}
```

will update field value with modifier "personal", but leave fields with other modifiers untouched. Result will be:

```
{
    "fields": {
        "first name": [{
            "value": "Jack",
            "modifier": ''
        }],
        "email": [{
            "value": "user@nimble.com",
            "modifier": "work"
        }, {
            "value": "user@nimble.com",
            "modifier": "personal"
        }]
    }
}
```

With `replace` parameter set to 1 if contacts that has:

```
{
    "fields": {
        "first name": [{
            "value": "Jack",
            "modifier": ''
        }],
        "email": [{
            "value": "user@nimble.com",
            "modifier": "work"
        }, {
            "value": "jack@gmail.com",
            "modifier": "personal"
        }]
    }
}
```

and then UPDATE with:

```
{
    "fields": {
        "email": [{
            "value": "user@nimble.com",
            "modifier": "personal"
        }]
```

```
    }
}
```

will replace `email` fields with all modifiers. Result will be:

```
{
    "fields": {
        "first name": [{
            "value": "Jack",
            "modifier": ''
        }],
        "email": [{
            "value": "user@nimble.com",
            "modifier": "personal"
        }]
    }
}
```

`fields` and `avatar_url` parameters are passed as JSON in request body. You should pass at least one of the parameters: `fields` or `avatar_url` (or both).

fields Describes a dictionary organized in the same structure as a contact listing response. In this structure, each key is field name. Values are lists of dicts, having two fields: value - actual value to store in contact field, modifier - field modifier to use, if field can have one. Values provided in this list will replace actual field's values for contact. If you want to remove all values from field — pass `null` as value.

avatar_url — optional, default: None String, pointing to avatar, that should be assigned to the contact.

Note: Nimble uses lazy loading mechanism for avatars, and didn't perform any checks for URL validness during `avatar_url` setting. If you'll pass invalid parameter here — no avatar will be displayed for contact.

Example:

```
{
    'fields': {
        'first name': [{
            'value': 'Jack',
            'modifier': ''
        }],
        'last name': [{
            'value': 'Daniels',
            'modifier': ''
        }],
        'phone': [{
            'value': null,
            'modifier': 'work'
        }]
    }
}
```

Response: OK

Updated contact is returned and encoded in the same way that is used in contacts listings.

```
{
    'updated': '2012-11-07T16:50:04+0200',
    'created': '2012-11-07T16:50:04+0200',
    'fields': {
        'last name': [{
```

```
            'field_id': '5049f697a694620a07000045',
            'modifier': '',
            'group': 'Basic Info',
            'value': 'Daniels',
            'label': 'last name'
        }],
        'source': [{
            'field_id': '5049f697a694620a0700004f',
            'modifier': '',
            'group': 'Basic Info',
            'value': 'm',
            'label': 'source'
        }],
        'first name': [{
            'field_id': '5049f697a694620a07000043',
            'modifier': '',
            'group': 'Basic Info',
            'value': 'Jack',
            'label': 'first name'
        }]
    },
    'object_type': 'contact',
    'id': '509a751c262b37af05000011',
    'last_contacted': {
        'last_contacted': null,
        'thread_id': null,
        'message_id': null
    },
    'record_type': 'person',
    'creator': 'Nimble API test',
    'children': [],
    'tags': [],
    'owner_id': '5049f696a694620a0700001c'
}
```


For more details see: *Contact resources*.

Response: Errors

Possible errors:

- *Validation Error*
- *Quota Error*
- *NotFound Error*

4.1.6 Delete contact

Simple contacts delete

Performs deletion of contacts by their ids.

Request

Example:

```
DELETE https://api.nimble.com/api/v1/contact/<id>,<id>,<id>...
```

Parameters

None, all contact's IDs specifies in URL, separated by comma

Response: OK

Request returns HTTP code 200, body contains OK status and list of IDs of deleted contacts.

Example:

```
{
    "status": "ok",
    "data": {
        "ids": [
            "5049f697a694620a0700007f",
            "5049f697a694620a07000082",
            "5049f697a694620a07000045"
        ]
    }
}
```

Response: Errors

Possible errors:

- *Validation Error*

Advanced contacts delete

Allows bulk removal of contacts by keyword or advanced search query.

Request

Example:

```
DELETE https://api.nimble.com/api/v1/contacts/list
```

Parameters

Parameters are similar to contact's listing ones.

keyword Delete all contacts where fields are containing value from this parameter

record_type — default: `all` Delete all contacts with provided record_type (`person` or `company`). This parameter could be combined with keyword parameter in order to delete contacts of specific record_type

query Json-encoded advanced search query to find contact for deletion. For more details on query syntax, see *Advanced search query syntax*.

> **Note:** If `query` parameter presented in request — `record_type` parameter will be ignored.

limit — default: `all` Number of contacts to delete by specified criteria (`query`, `keyword`, `record_type`).

Warning: `query` and `keyword` parameters are mutually exclusive. If you'll try to specify both — validation error will be returned.

Response: OK

Example request:

```
DELETE https://api.nimble.com/api/v1/contacts/list?keyword=DoeAPItest
```

Response will be:

```
{
    "status": "ok",
    "data": {
        "ids": [
            "50941746837d4e3df20001d1",
            "50941746837d4e3df1000144"
        ]
    }
}
```

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error*

4.2 Contact Notes API

Contents:

4.2.1 Show single note

Request

Base endpoint:

```
GET https://api.nimble.com/api/v1/contacts/notes/<note_id>
```

Parameters

note_id

Single note id to show

```
GET https://api.nimble.com/api/v1/contacts/notes/508a4750084abd28bc00016f
```

Response: OK

Response to this request is dictionary with JSON serialised note body.

Contact note


```
{
    "created": "2012-11-29T10:16:46+0000",
    "contacts": [{
        "id": "508a4750084abd28bc00016f",
        "name": "Jack Daniels"
    }],
    "note_preview": "note 1",
    "author_name": "Nimble API test",
    "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica%2C
↪%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
    "id": "50b7360e837d4e4404000013",
    "owner_id": "5049f696a694620a0700001c"
}
```


Keys meaning:

id Id of the note. Can be used to identify it in update and delete operations.

created Date and time when the note was created encoded in ISO 8601

contacts List of `id-name` pairs for each of contacts associated with the note

note_preview Note content with all formatting removed

note Note content with all formatting mark up in place

author_name Readable name of company user who created this note

owner_id Id of company user who created this note

Response example


```
{
    "created": "2012-11-29T10:16:46+0000",
    "contacts": [{
        "id": "508a4750084abd28bc00016f",
        "name": "Jack Daniels"
    }],
    "note_preview": "note 1",
    "author_name": "Nimble API test",
    "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica%2C
↪%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
    "id": "50b7360e837d4e4404000013",
    "owner_id": "5049f696a694620a0700001c"
}
```


Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error*

4.2.2 Get contact notes list

Request

Base endpoint:

```
GET https://api.nimble.com/api/v1/contact/<contact_id>/notes
```

Example:

GET https://api.nimble.com/api/v1/contact/5049fba29b85f669e40004fb/notes

Parameters

All parameters are optional. Unrecognized parameters are ignored. Unrecognized values will return an error.

per_page — default: 5

Specifies the number of items to return per page of results.

page — default: 1

Specifies which page to display. Numeration starts from 1.

Response: OK

Response to this request is dictionary formed out of two keys: `meta` and `resources`. Metadata coming under `meta` key is common structure used among different listing in Nimble. Value under `resources` key is list of notes entries.

Pagination metadata

```
'meta': {
    'per_page': 30,
    'total': 45,
    'pages': 2,
    'page': 1
}
```

Keys meaning:

per_page Number of contacts returned per page

total Total number of contacts

pages Total pages count

page Current page number

Contact note

```
{
    "created": "2012-11-29T10:16:46+0000",
    "contacts": [{
        "id": "508a4750084abd28bc00016f",
        "name": "Jack Daniels"
    }],
    "note_preview": "note 1",
    "author_name": "Nimble API test",
    "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica%2C
↪%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
    "id": "50b7360e837d4e4404000013",
    "owner_id": "5049f696a694620a0700001c"
}
```

Keys meaning:

id Id of the note. Can be used to identify it in update and delete operations.

created Date and time when the note was created encoded in ISO 8601

contacts List of `id-name` pairs for each of contacts associated with the note

note_preview Note content with all formatting removed

note Note content with all formatting mark up in place

author_name Readable name of company user who created this note

owner_id Id of company user who created this note

Response example


```
{
    "meta": {
        "per_page": 5,
        "total": 1,
        "pages": 1,
        "page": 1
    },
    "resources": [{
        "created": "2012-11-29T10:16:46+0000",
        "contacts": [{
            "id": "508a4750084abd28bc00016f",
            "name": "Jack Daniels"
        }],
        "note_preview": "note 1",
        "author_name": "Nimble API test",
        "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica
↪%2C%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
        "id": "50b7360e837d4e4404000013",
        "owner_id": "5049f696a694620a0700001c"
    }]
}
```


Response: Errors

Possible errors:

- *Validation Error*

4.2.3 Create contact note

Request

Base endpoint:

```
POST https://api.nimble.com/api/v1/contacts/notes
```

Parameters

All parameters are passed as JSON in request body. All parameters are mandatory.

contact_ids

List of contacts' IDs in BSON format to which the note will be attached. Contacts count should be between 1 and 10.

note

Sting, containing note itself.

note_preview

Short version of note, that will be used for preview purposes.

Example:

```
{
    "contact_ids": ["50c07a69e5ef834edb000080", "50c07a53084abd5f61000aac"],
    "note": "Just contact note, with some longer text",
    "note_preview": "Just contact note"
}
```

Response: OK

Response to this request is dictionary with JSON serialised note body.

Contact note

```
{
    "created": "2012-11-29T10:16:46+0000",
    "contacts": [{
        "id": "508a4750084abd28bc00016f",
        "name": "Jack Daniels"
    }],
    "note_preview": "note 1",
    "author_name": "Nimble API test",
    "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica%2C
↪%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
    "id": "50b7360e837d4e4404000013",
    "owner_id": "5049f696a694620a0700001c"
}
```

Keys meaning:

id Id of the note. Can be used to identify it in update and delete operations.

created Date and time when the note was created encoded in ISO 8601

contacts List of `id-name` pairs for each of contacts associated with the note

note_preview Note content with all formatting removed

note Note content with all formatting mark up in place

author_name Readable name of company user who created this note

owner_id Id of company user who created this note

Response example

```
{
    "created": "2012-11-29T10:16:46+0000",
    "contacts": [
        {
            "id": "508a4750084abd28bc00016f",
            "name": "Jack Daniels"
        }
    ],
```

```
    "note_preview": "note 1",
    "author_name": "Nimble API test",
    "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica%2C
↪%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
    "id": "50b7360e837d4e4404000013",
    "owner_id": "5049f696a694620a0700001c"
}
```

Response: Errors

Possible errors:

- *Validation Error*

4.2.4 Update contact note

Request

Base endpoint:

```
PUT https://api.nimble.com/api/v1/contacts/notes/<note_id>
```

Parameters

All parameters are passed as JSON in request body. All parameters are mandatory.

contact_ids

List of contacts' IDs in BSON format to which the note will be attached. Contacts count should be between 1 and 10.

note

Sting, containing note itself.

note_preview

Short version of note, that will be used for preview purposes.

Example:

```
{
    "contact_ids": ["50c07a69e5ef834edb000080", "50c07a53084abd5f61000aac"],
    "note": "Just contact note, with some longer text",
    "note_preview": "Just contact note"
}
```

Response: OK

Response to this request is dictionary with JSON serialised note body.

Contact note

```
{
    "created": "2012-11-29T10:16:46+0000",
    "contacts": [{
        "id": "508a4750084abd28bc00016f",
```

```
        "name": "Jack Daniels"
    }],
    "note_preview": "note 1",
    "author_name": "Nimble API test",
    "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica%2C
↪%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
    "id": "50b7360e837d4e4404000013",
    "owner_id": "5049f696a694620a0700001c"
}
```

Keys meaning:

id Id of the note. Can be used to identify it in update and delete operations.

created Date and time when the note was created encoded in ISO 8601

contacts List of `id-name` pairs for each of contacts associated with the note

note_preview Note content with all formatting removed

note Note content with all formatting mark up in place

author_name Readable name of company user who created this note

owner_id Id of company user who created this note

Response example

```
{
    "created": "2012-11-29T10:16:46+0000",
    "contacts": [{
        "id": "508a4750084abd28bc00016f",
        "name": "Jack Daniels"
    }],
    "note_preview": "note 1",
    "author_name": "Nimble API test",
    "note": "%3Cfont%20face%3D%22Arial%2C%20Tahoma%2C%20Verdana%2C%20Helvetica%2C
↪%20sans-serif%22%3Enote%201%3C%2Ffont%3E",
    "id": "50b7360e837d4e4404000013",
    "owner_id": "5049f696a694620a0700001c"
}
```

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error*

4.2.5 Delete single note

Request

Base endpoint:

```
DELETE https://api.nimble.com/api/v1/contacts/notes/<note_id>
```

Parameters

note_id

Single note id to delete

```
DELETE https://api.nimble.com/api/v1/contacts/notes/508a4750084abd28bc00016f
```

Response: OK

Response to this request is dictionary with id of deleted note.

Response example

```
{
    "id": "508a4750084abd28bc00016f"
}
```

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error*

4.3 Contacts Metadata API

Contents:

4.3.1 List all contacts metadata (fields and groups)

Request

API endpoint:

```
GET https://api.nimble.com/api/v1/contacts/metadata
```

Response: OK

Call to this API endpoint will return *metadata information*.

4.3.2 Create field

Request

Example:

```
POST https://api.nimble.com/api/v1/contacts/metadata/fields
```

Parameters

All parameters are passed as JSON in request body. All parameters are mandatory.

name

name for new field.

> **Note:** Name should be unique.

group_id

id of fields group, new field should belong to.

presentation

dictionary describing how field should be presented in Nimble client (can be empty dictionary as well). More details are at *described here*.

Example:

```
{
    "group_id": "5092a4d5084abd46de000725",
    "presentation": {
        "width": "1",
        "type": "single-line-text-box"
    },
    "name": "new field"
}
```

Response: OK

On success, server returns response with HTTP code 201 and, newly created, encoded field.

```
{
    "group": "Some new tab",
    "name": "new field",
    "label": "new field",
    "modifier": "",
    "presentation": {
        "width": "1",
        "type": "single-line-text-box"
    },
    "id": "50cf3eca084abd0f070013ae",
    "multiples": false
}
```

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error* (in case of invalid value in *group_id* parameter).

4.3.3 Update field

Request

Example:

```
PUT https://api.nimble.com/api/v1/contacts/metadata/fields/<field_id>
```

Parameters

field_id

id of field we want perform an update for.

Other parameters are passed as JSON in request body. All parameters are mandatory.

name

name for new field.

Note: Name should be unique.

---

group_id

id of fields group, new field should belong to.

presentation

dictionary describing how field should be presented in Nimble client (can be empty dictionary as well). More details are at *described here*.

Example:

```
{
    "group_id": "5092a4d5084abd46de000725",
    "presentation": {
        "width": "1",
        "type": "single-line-text-box"
    },
    "name": "new field2"
}
```

Response: OK

On success, server returns response with HTTP code 200 and, recently updated, encoded field.

```
{
    "group": "SOme new tab",
    "name": "new field2",
    "label": "new field2",
    "modifier": "",
    "presentation": {
        "width": "1",
        "type": "single-line-text-box"
    },
    "id": "50cf3eca084abd0f070013ae",
    "multiples": false
}
```

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error* (in case of invalid value in *group_id* or *field_id* parameter).

4.3.4 Delete field

Request

Example:

```
DELETE https://api.nimble.com/api/v1/contacts/metadata/fields/<field_id>
```

Parameters

field_id

id of field we want to delete.

Note: Only custom fields are allowed to be deleted.

Other (single one) parameters are passed as JSON in request body.

force

boolean, this parameter indicated whether we want to remove field even if it has values across some contacts.

If this parameter omitted in request it will counted as false.

Example:

```
{
    "force": true,
}
```

Response: OK

On success, server returns empty response with HTTP code 200.

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error* (in case of invalid value in *field_id* parameter).

4.3.5 Create fields group

Request

Example:

```
POST https://api.nimble.com/api/v1/contacts/metadata/groups
```

Parameters

All parameters are passed as JSON in request body. All parameters are mandatory.

type

type of fields group (only for contacts persons, only for companies or for both). Possible types are: *person*, *company*, *both*.

name

name for new fields group.

Note: Name should be unique.

Example:

```
{
    "type": "both",
    "name": "grp123"
}
```

Response: OK

On success, server returns response with HTTP code 201 and, newly created, encoded fields group.

```
{
    "name": "grp525496_m",
    "label": "grp525496_m",
    "is_standard": false,
    "order": [],
    "type": "both",
    "id": "50cf3ecce5ef833f4f000341"
}
```

Response: Errors

Possible errors:

- *Validation Error*

4.3.6 Update fields group

Request

Example:

```
PUT https://api.nimble.com/api/v1/contacts/metadata/groups/<group_id>
```

Parameters

group_id

id of fields group we want to perform update for

All parameters are passed as JSON in request body.

type

type of fields group (only for contacts persons, only for companies or for both). Possible types are: *person*, *company*, *both*.

name

mame of new fields group.

Note: Name should be unique.

---

order

list, contains names of fields from group we try to perform update for in needed order.

If this parameter omitted in request it will counted as empty list.

Example:

```
{
    "type": "both",
    "name": "grp525496_m2",
    "order": []
}
```

Response: OK

On success, server returns response with HTTP code 200 and, recently updated, encoded fields group.

```
{
    "name": "grp525496_m2",
    "label": "grp525496_m2",
    "is_standard": false,
    "order": [],
    "type": "both",
    "id": "50cf3ecce5ef833f4f000341"
}
```

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error* (in case of invalid value in *group_id* parameter).

4.3.7 Delete fields group

Request

Example:

```
DELETE https://api.nimble.com/api/v1/contacts/metadata/groups/<group_id>
```

Parameters

group_id

id of group we want to delete.

Note: Only custom groups are allowed to be deleted.

Other (single one) parameters are passed as JSON in request body.

force

boolean, this parameter indicated whether we want to remove group even if it has some fields within itself.

If this parameter omitted in request it will counted as false.

Example:

```
{
    "force": false,
}
```

Response: OK

On success, server returns empty response with HTTP code 200.

Response: Errors

Possible errors:

- *Validation Error*
- *NotFound Error* (in case of invalid value in *group_id* parameter).

CHAPTER 5

Full description of Nimble default contact fields

5.1 Field types

Here described basic field types in Nimble. For full list of Nimble default fields — *see below*.

5.1.1 Default fields

Simple text fields, like `first name`, `last name`, `title`, `description`, etc.

5.1.2 Address

All values represented as dictionary with following keys: `street`, `city`, `state`, `zip`, `country`. This dictionary should be dumped to JSON string, and this string should be used as field's value.

Example:

```
{
    "type": "person",
    "fields": {
        "address": [{
            "value": "{\"street\":\"Test\", \"city\":\"Testing\", \"country\":\
↪"Togo\"}",
            "modifier": "other"
        }]
    }
}
```

5.1.3 Social fields

For creation of contacts with social fields, all field values should correspond specific rules.

Twitter Value should be twitter's username, e.g. `nimble` or `twitter`

Facebook Value should be users's Facebook profile URL, e.g. `http://www.facebook.com/grigori.rasputin`

LinkedIn Value should be users's LinkedIn profile URL, e.g. `http://ua.linkedin.com/in/grigori.rasputin`

Google+ Value should be users's Google+ profile URL, e.g. `https://plus.google.com/265456261827029907830/`

Fousquare Value should be: user's id in Foursquare, user's screen name (if set) or Foursquare profile URL.

In response, for every contact, Nimble adds additional information to fields, fetched from social network:

avatar_url URL of users's avatar in this social network.

user_id Network-specific unique ID of user's account.

user_name User's name, obtained from social account.

Warning: To correctly get data from social networks, user should connect appropriate social network account in Nimble. If no account connected — Nimble sometimes could be not able to fetch data.

Example:

```
"facebook": [{
    "avatar_url": "http://graph.facebook.com/210857648102/picture",
    "group": "Contact Info",
    "user_id": "210857648102",
    "user_name": "Nimble",
    "modifier": "",
    "field_id": "4eabb2494fb88d3352011a82",
    "value": "http://www.facebook.com/nimble",
    "label": "facebook"
}]
```

5.1.4 Parent company

Usual string, representing parent company for this person's contact. If company with corresponding name (search are case-insensitie) not found — it will be created. Value record for this field contains additional key `extra_value`, holding unique ID of parent company.

5.1.5 Domain

The domain field. Example: `nimble.com`. This field satisfies the following conditions:

- Unique in terms of team. It is possible to have only one company record with particular value in the whole account.
- Properly formatted. No protocol or path is allowed, it can have 3rd level domain at most.
- It can be assigned to company records only

5.1.6 Dropdown fields

Fields, showing as drop-down lists in Nimble. In metadata they have `field_type` equal `select-box`. Also, their metadata contains field `values`, representing drop-down content. This field contains list of dictionaries, having two keys:

id Value, to be stored in field

value String, corresponding to this value

Example:

```
"lead status": [{
    "field_type": "select-box",
    "group": "Lead Details",
    "label": "lead status",
    "values": [{
        "id": "1",
        "value": "Open"
    }, {
        "id": "2",
        "value": "Contacted"
    }, {
        "id": "3",
        "value": "Qualified"
    }, {
        "id": "4",
        "value": "Unqualified"
    }],
    "modifier": "",
    "id": "5049f697a694620a0700008d"
}]
```

5.2 Nimble default fields

Table 5.1: Full list of Nimble default fields

Field Name	Type	Multiple field	Modificators	Notes
first name	*default*	-	N/A	For person contact
last name	*default*	-	N/A	For person contact
middle name	*default*	-	N/A	
company name	*default*	-	N/A	For company contact
title	*default*	-	N/A	
parent company	*parent company*	-	N/A	
domain	*domain*	-	N/A	Unique. For company contact
source	*default*	-	N/A	Source of this contact (import, manual creation, etc.)
last contacted	outdated	-	N/A	Replaced by corresponding field in contact resource
phone	*default*	+	• work • home • mobile • main • home fax • work fax • other	
email	*default*	+	• work • personal • other	
skype id	*default*	+	N/A	
twitter	*social*	+	N/A	
facebook	*social*	+	N/A	
linkedin	*social*	+	N/A	
google plus	*social*	+	N/A	
foursquare	*social*	+	N/A	
address	*address*	+	• work • home • other	
hubspot	*default*	-	N/A	
URL	*default*	+	• work • personal • blog • other	
description	*default*	+	• other • twitter • facebook • linkedin • google+ • foursquare	If possible, fetches descriptions from social networks
annual revenue	*default*	-	N/A	

5.3 Nimble default field groups

Table 5.2: Nimble default field groups

Group Name	Description	Fields
Basic info	Contact's basic info	• first name, • last name, • middle name, • company name, • title, • parent company, • source, • last contacted
Personal Info	Personal contact's details	• birthday
Extra Info	Contact's extended information	• URL, • description
Contact Info	How to reach this contact	• phone, • email, • skype id, • twitter, • facebook, • linkedin, • google+, • foursquare, • address, • hubspot
Company Info	Extended information about contact's company	• annual revenue, • # of employees
Lead Details	Information abut contact as lead	• lead status, • rating, • lead source, • lead type,

5.4 Nimble fields presentation

To control, how contacts will look in Nimble, special parameter `presentation` included in fields metadata. Usually it is a dictionary with 2 fields:

- **type** — represents type of field in nimble. It can have one of the following values:
  - single-line-text-box — simple field with one line of text
  - multi-line-text-box — field, containing multiline text
  - select-box — drop-down list with predefined values, require additional parameter `values`, containing list of dictionaries, representing list items (see examples below)
  - separator — separator with heading, that could be used for logical fields grouping

  - address — field with address, that will allow input of address in Nimble default format

- **width** — integer value of 1 or 2, represents, how many columns will this field occupy in Nimble.

Examples:

```
{
    width: 1,
    type: "single-line-text-box"
}
```

```
{
    width: 1,
    values: [{
        id: 1,
        value: "Yes"
    }, {
        id: 2,
        value: "No"
    }, {
        id: 3,
        value: "I don't know"
    }],
    type: "select-box"
}
```

CHAPTER 6

Activities API

Contents:

6.1 Tasks API

Contents:

6.1.1 Create task

Request

Base endpoint:

```
POST https://api.nimble.com/api/v1/activities/task
```

Parameters

All parameters are passed as JSON in request body and could be omitted except *subject*.

subject — mandatory

Subject (short description) of new task. Field length should be between 2 and 128 chars.

notes

Any additional text (notes) for task.

related_to

List with contact ids we want to bind this task with.

due_date

Due date for new task. Should be passed in format like "YYYY-MM-DDTHOURS:MINUTES:SECONDS". Example: "2013-04-04T13:50:00"

Example:

```
{
    "due_date": "2013-04-04T13:50:00",
    "notes": "Blah blah blah blah \u0440\u0443\u0441\u0441\u043a\u0438\u0439␣
↪\u0442\u0435\u043a\u0441\u0442 8168949",
    "related_to": [
        "508a4750084abd28bc00016f"
    ],
    "subject": "Hello task! 2423056"
}
```

Response: OK

Response to this request is dictionary with JSON serialised task body.

Response example

```
{
    "related_to": [
        "508a4750084abd28bc00016f"
    ],
    "due_date": "2013-04-04T13:50:00",
    "notes": "Blah blah blah blah \u0440\u0443\u0441\u0441\u043a\u0438\u0439␣
↪\u0442\u0435\u043a\u0441\u0442 8168949",
    "id": "5108f1cc837d4e3930e297fb",
    "subject": "Hello task! 2423056"
}
```

Response: Errors

Possible errors:

- *Validation Error*

CHAPTER 7

Tutorial: Obtain Nimble API Key

Contents

7.1 Introduction

This document will describe authorization flow that return the authorization token that allow to make requests to Nimble API and access resources to which access has been granted for user. For For authorization Nimble use OAuth 2.0 protocol with bearer tokens. All technical details can be read in RFC6749 and RFC6750, but you don't need to read them (unless you want to know more details on OAuth 2.0) — every detail that need to obtain access token and use this token for requesting resource on behalf of user will be described in this document.

7.2 Terminology

User Person who has an account inside Nimble and owner of resources to which API provide access

Client Service that request a token and want to make requests to the Nimble API on behalf of User

Authorization server Server that allow User to grant access for Client to use his resources

Resource server Server that returns User's resources to Client if it was granted for access by User

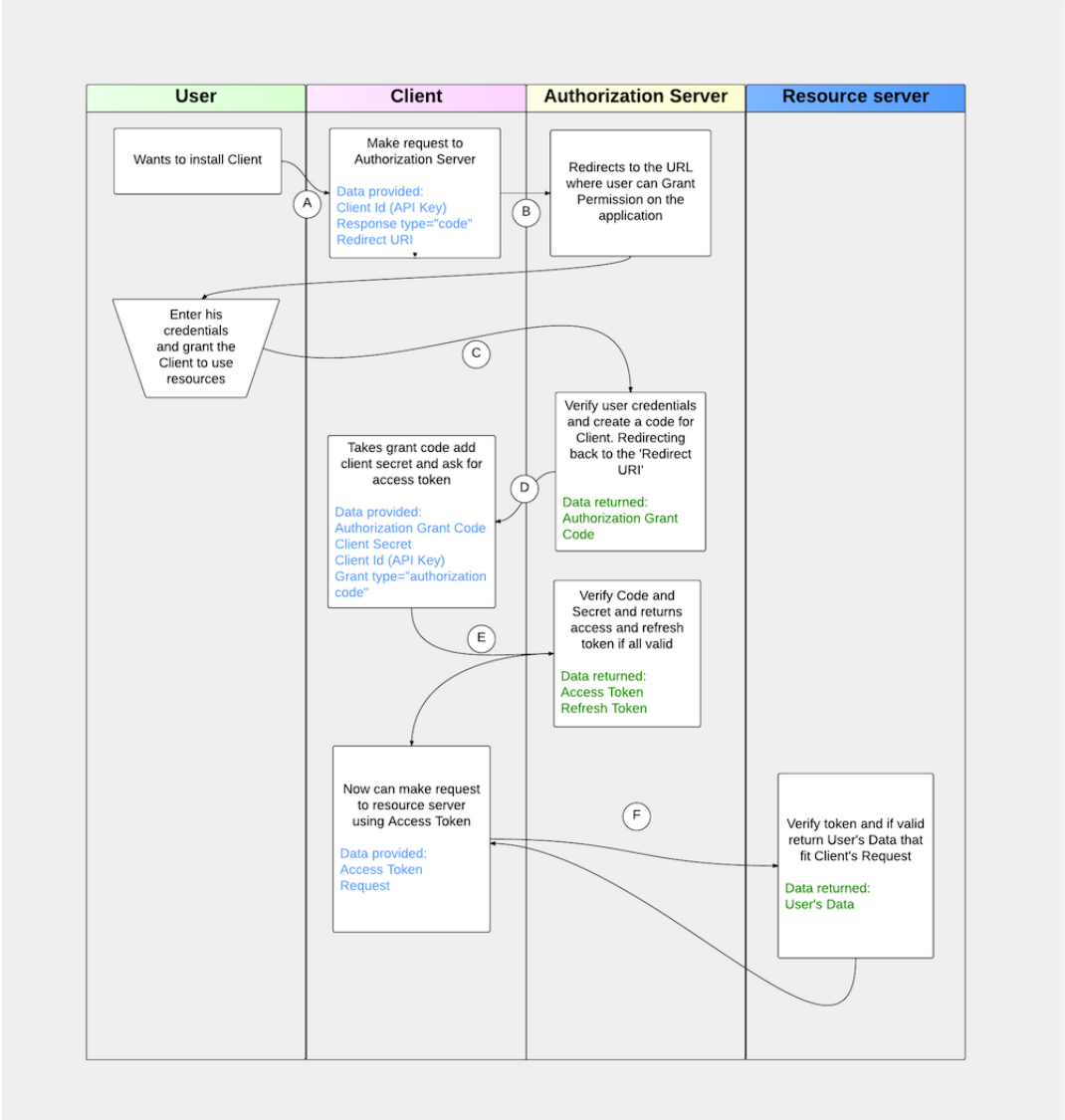
Bearer token Token that Client received after User has granted access. Possession of this token allows client to make requests to Resource server in order to receive Client's resources.

7.3 Prerequisites

In order to create a Client for Nimble API you need to have to register your application in our DB and get Client Secret and Client Id. Its can be done on the page with this link: https://support.nimble.com/third-party-integrations/nimble-api-access/nimble-api-access. You probably have done this already.

7.4 Authorization process overview

For giving you a good overview of process we will use scheme and give a brief explanation to the each step on scheme. Steps are shown by the arrows and has a letters assigned to each step on scheme.



Let's go over a scheme step by step.

- **Step A** — You have some product (Client) that wants to use Nimble Data for some purposes. User wants to use Client and you need to retrieve a grant of user to operate the data from Nimble on his behalf. So, user has something that allows to initiate this process from Client.
- **Step B** — Your client open a page that points to Authorization Server providing a params specified on scheme. As a params, you need to send your API key and URI of client to which Authorization Grant Code will be returned. For details see: *Requesting Authorization Grant Code*.
- **Step C** — Auth server using your API Key (Client Key) creates a link to which user will be directed and send a redirect response. User will be automatically redirected to the page where he can put his credentials and grant access. Note that now user is on side of Authorization Server.
- **Step D** — As soon as user finished a process of providing access, Authorization Server takes a redirect URI that you specified on step B and sends code to that URI. Response details *explained here*.
- **Step E** — Using this code you make a *application/x-www-form-urlencoded* request to the Authorization Server where in body you put code retrieved on previous step, your Client Secret, Client Id and Grant Type you want to receive. It is always *authorization_code*. For details see: *Requesting Access Token*.
- **Step F** — If everything is valid then you will receive response from server with Access Token and Refresh Token. Now you are able to do a requests to the Nimble API using Access Token until it valid. Using Refresh Token you will be able to renew this token without involving User second time. For details see: *Refresh token after expiration without user input*.

7.5 Requesting Authorization Grant Code

You should use this request on step B of *Authorization Process*. You need to open a page for user with this endpoint and provide a Redirect URI on which your handler will be able to catch the code from Authorization Server that will be returned when User successfully grant you a permission to use Nimble API.

Endpoint:

```
GET https://api.nimble.com/oauth/authorize
```

Params:

client_id *required* — Your Client Key from Application Page.

redirect_uri *required* — URI where you have a handler who will catch a code and finish the Process, see note below.

response_type *required* — must be set to `code`. We don't support Implicit Flow, so `code` is the only available option now.

scope *optional* — for now there is only one scope for Nimble API, so skip this parameter for now.

Note: Please note, that main value for redirect URL is specified in application settings on developer portal. `redirect_uri` parameter in URL could be used only to overwrite path part in redirect URL. So, `redirect_uri` should have exactly same URI, as specified in application settings.

Example request:

```
GET https://api.nimble.com/oauth/authorize?client_id=5f96b5e9adaxzca93x1213123132&
↪redirect_uri=https%3A%2F%2Fyourportal.com%2Fauth%2Fpassed&response_type=code
```

Successful response:

First, user will be redirected to the page on Authorization Server with hostname `https://api.nimble.com/oauth/authorize`

As soon as he provided his credentials, you will receive a request like listed below on your `Redirect URI`:

```
https://yourportal.com/auth/passed?code=LTM4M
```

Error response:

If the request is missing or has incorrect parameters, the user-agent will be redirected back to the redirect URI provided. The redirection will contain parameters specifying the error.

Example Invalid Authorization Request Redirect:

```
http://www.myapp.com/oauth?error=invalid_request&error_description=Invalid
↪%20URL
```

After selecting Login, the user will be validated. If user validation is successful, a consent page is displayed. If user validation is unsuccessful, the user-agent will be redirected to the redirect URI provided in the initial request. This redirection will include additional parameters specifying the error.

Example Unsuccessful Validation Redirect:

```
http://www.myapp.com/oauth?error=access_denied&error_descripton=Validation
↪%20errors
```

If user click Deny on the grant permission page then another error will be sent.

Example Deny Consent Redirect:

```
http://www.myapp.com/oauth?error=access_denied&error_description=User
↪%20denied%20access
```

7.6 Requesting Access Token

As soon as User complete step C your handler will catch step D. You need to listen for redirect on your Redirect URI. **Code returned to you isn't access token yet!** You still need to obtain the authorization token. Note, that this code is valid for a short period time and if you not intiate request to access token as soon as you receive a code then received code can become invalid and User will need to reinitiate a process once again. So, on step E you need to receive access to token for which user granted you.

The Client should use the authorization code obtained to request an access token. When requesting an access token, you SHOULD specify required data as form parameters. Client application secret is needed for client authentication. When specifying client_id and client_secret as form parameters, the `Content-Type` header MUST be set to `application/x-www-form-urlencoded`. Request should be done via HTTPS only.

Endpoint:

```
POST https://api.nimble.com/oauth/token
```

Parameters:

grant_type *required* — must be set to `authorization_code`. You need to receive an Access token.

code *required* — code that you received on step D. This code has a short-valid time, so initiate request for token as soon as you receive it.

redirect_uri *required* — redirect URI for your application. Should be equal to `redirect_uri`, provided during *Requesting Authorization Grant Code*.

client_id *required* — your Client API key.

client_secret *required* — your Client API secret key.

Headers:

Content-Type: application/x-www-form-urlencoded; charset=UTF-8
required — you need to specify this header always

Example Request:

```
POST /oauth/token HTTP/1.1
Host: api.nimble.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

Body : client_id=5f96b5e9a6b7478e15ee574a426aa063&redirect_uri=http%3A%2F
↪%2Flocalhost%3A3000%2Fauth&code=LTM4M&grant_type=authorization_code&client_
↪secret=89bb4ffb4f264bff
```

Successfull Response JSON:

```
{
    "access_token": "bf086611-9e97-4d11-9cd7-3c86dec0bbd4",
    "token_type": "bearer",
    "expires_in": 599,
    "refresh_token": "515ac59b-6518-49a2-81d6-54f91ee74c4a",
    "scope": "read write"
}
```

7.7 API requests using Access Token

Now when we have Access Token Received you need to store it and use for any requests for Nimble Data on behalf of user. This process described in *second part of our tutorial*.

7.8 Refresh token after expiration without user input

The application uses the refresh token to extend the validity of the access token provided with the refresh token. When refreshing an access token, you should specify required data as a form parameters. Client application secret is needed for client authentication. `Content-Type` header must be set to `application/x-www-form-urlencoded`.

Parameters:

client_id Client identifier used to obtain the authorization code

client_secret Client secret code

grant_type Must be set to `refresh_token`

refresh_token Refresh token obtained from the access token request

redirect_uri *required* — redirect URI for your application. Should be equal to `redirect_uri`, provided during *Requesting Authorization Grant Code*.

Example Request:

```
POST /oauth/token HTTP/1.1
Host: https://api.nimble.com/
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

client_id=3e8471e7516a0c85ef35ab1d23f1bdf1&client_secret=737d10deba3fd124&grant_
↪type=refresh_token&refresh_token=5f752714eddb07a3e41c2a3311f514e1&redirect_
↪uri=http%3A%2F%2Flocalhost%3A3000%2Fauth
```

Example Response:

```
{
    "access_token": "1d7bc7328b402f4826e17607e364bc6a",
    "expires_in": 559,
    "refresh_token": "f35c2165112fda74f79b408cc253485fcdfd888a"
}
```

7.9 Examples

For your convinience we created some examples:

Python authorization example. Actual code implementation on Python and Tornado

Ruby authorization example. Implementation of authorization process in Ruby

7.10 Troubleshooting & Feedback

If you have any problems or want to submit feedback feel free to go to our support forum or email us at api-support@nimble.com

CHAPTER 8

Tutorial: Making authenticated requests

When you've received token, using the process described *here* you are ready to call Nimble API. You can use one of two ways, described below. They both are equal.

8.1 Authenticating your request with URL parameter

In order to use this token code you just add it into URL as request parameter with name `access_token`.

Endpoint:

Any of available endpoint of Nimble API

Params:

No matter what request `POST`, `GET` or any other HTTP method, just add an `access_token` as parameter to URL.

access_token *required* — put a token for user under this parameter.

Example Request:

```
POST https://api.nimble.com/api/v1/contacts?access_
↪token=e0f7b053200672c2ff6ede59c8e2bfc7
```

Successful Response:

All API responses described on their corresponding pages.

8.2 Authenticating your request with HTTP header

You can also pass your token in HTTP header `Authorization` in format: `Bearer <your token>`.

Example request:

```
PUT /api/v1/contact/4f60a873fcf7b752ed006b7a HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate, compress
Authorization: Bearer c0b5f46631455b543c309b8cb18b8dae
Content-Type: application/json; charset=utf-8
```

```
{
    "fields": {
        "first name": [
            {
                "modifier": "",
                "value": "1name"
            }
        ]
    }
}
```

// TODO specify invalid token or expired token error

CHAPTER 9

Examples and libraries

Here we collected list of exa

9.1 Javascript

Javascript example for Nimble REST API

https://github.com/nimblecrm/javascript-example

Requires PHP for proxying OAuth login, uses jQuery to fetch contact list via JSONP requests.

9.2 NodeJS

node-nimble-api

https://npmjs.org/package/node-nimble-api

Javascript wrapper for Nimble CRM API

9.3 Python

Python simpe example for Nimble API

https://github.com/nimblecrm/python-example

Simple python client, demonstrating basic usage of Nimble API. Requires Tornado. Includes Nimble OAuth mixin, ready to use in your applications.

9.4 Ruby

Omniauth::Nimble gem

https://github.com/nimblecrm/omniauth-nimble

Allows to use OmniAuth gem with Nimble.

Simple Ruby client for Nimble API

https://github.com/nimblecrm/ruby-example

Two basic examples, showing Nimble API usage with Ruby.

CHAPTER 10

Nimble API changes history

Version 1.3

- *New fields* available for custom search

Version 1.2

- *New fields* available for custom search

Version 1.1

- Added *basic activities API*

Version 1.0

- Our API goes public
- Added *contact notes API*

Version 0.9

- Added *contact metadata API*

Version 0.8

- Added *CRUD operations for contacts*
- Added *Advanced Search*

Version 0.7

- First private beta version. *Contacts listing and simple search*

CHAPTER 11

Getting help

If you need our assistance or want to share some comments/suggestions — please feel free to contact us via email api-support@nimble.com.

Your feedback is greatly appreciated while we continue to shape our API offering.