Limfinity

API Guide

RURO, Incorporated

321 Ballenger Center Dr., Suite 102,

Frederick, MD 21703

USA

Phone: 1-888-881-7876

E-mail: info@ruro.com

www.ruro.com

Title:	**Limfinity 7 API Guide**
Doc No:	**DHF-Limfinity-6.5.0-API Guide-01**
Rev. No:	**3.0**

Revision Record

Revision Number	Date	Author	Description of Change
1.0	07/01/16	QA	Initial Release
2.0	6/20/17	QA	Updated Styles
3.0	8/27/19	Alex Nikolaitchik	Updated Content

Table of Contents

Section 1. List of Objects Supported by Limfinity API

Following is a list of Objects and their identifying values that are called by supported methods. For the corresponding methods see ‘Section 2. List of API Functions’.

1.1 AuditRec:

A record of any change within the system.

:id = Unique ID of an audit record.

:obj_name = Name of the changed object.

:user = The name of the user that made the change.

:obj_type = The type of the object changed.

:created_at = The date the audit record was created.

:message = The action performed that created the audit.

:comments = Any user-generated comment added when the audited action was performed.

1.2 Auth_token:

An authorization token used in place of a user’s password for API calls. Using the auth_token will omit the “API Session Created” and “API Session Removed” entries in the audit log.

1.3 Users:

Uniquely identifies a user in the system:

:**id** = Unique ID of a user.

:username = User’s Login name.

:fullname = User’s full name.

:email = User’s email.

:created_at = Date when the user was created in the system.

:roles = Roles assigned to the user.

:disabled = Defines whether the user is currently disabled.

:locked = Defines whether the user is currently locked.

:active = Defines whether the user is currently active.

:groups = List of User Groups this user belongs to.

1.4 Group:

Uniquely identifies a user group in the system:

:id = Unique ID of a user group.

:name = User group's name.

:description = Description of the group.

:created_at = Date when the group was created in the system.

:created_by = User name who created this group.

:updated_at = Date when the group was last updated.

:users-count = Number of users in the group.

:baseline = Indicates if the object is part of the baseline configuration.

1.5 Role:

A role that defines the access permissions in the system:

:id = Unique ID of a role.

:name = Role's name.

:rights = A list of rights assigned to the role.

:baseline = Indicator of the role being part of a baseline configuration.

:created_at = Date when the role was created in the system.

:system_role = Defines whether the role is internal (built-into the software) or user-defined.

1.6 UserField

A user-defined field in the system:

:id = Unique ID of a user-defined field.

:name = Internal name of the user-defined field.

:display_name = Display (human readable) name of user-defined field.

:type = User-defined field type ("Date", "Text Field", "Text Area", "Checkbox", etc.)

:searchable = Defines whether the user-defined field is searchable.

:values = User-defined field values.

:created_at = Date when the user-defined field was created in the system.

:updated_at = Date when the user-defined field was last updated.

:created_by = User name who created this user-defined field.

:used_by = Which objects use the user-defined field.

:permission = Permissions applicable to the user calling a method which returns this object.

1.7 SubjectType:

A user-defined subject type in the system:

:id = Unique ID of a subject type.

:name = Internal name of the subject type.

:descr = Description of the subject type.

:color = Color of the icon representing the subject type.

:searchable_quick = Defines whether the subject type may be found in a quick search.

:searchable_advanced = Defines whether the subject type may be found in an advanced search.

:searchable_batch = Defines whether the subject type may be found in a batch search.

:fields = List of user-defined field names.

:searchable= The subject type is searchable within the system.

:syslock = Indicates if the object is "locked" in the configuration, and can't be changed regardless of the site.

:baseline = Indicates if the object is part of the baseline configuration.

:created_at = Date when the subject type was created in the system.

:updated_at = Date when the subject type was last updated.

:created_by = User name who created this subject type.

:enabled = Enabled or disabled flag for this subject type.

:permission = The permissions available to this subject type.

:configuration = The subject type is part of the system configuration.

:fields_count = Number of user-defined fields in this Subject Type.

1.8 SubjectTypeGroup:

A user-defined group of subject types in the system:

:id = Unique ID of a subject type group.

:subject_types = The subject types associated with this group.

created_at = Date when the subject type group was created.

baseline = Indicates if the object is part of the baseline configuration.

updated_at = Date when the subject type group was last updated.

created_by: = User name who created this subject type group.

1.9 Subject:

Any user object in the system. Subjects are instances of a particular Subject Type:

:id = Unique ID of a subject.

:name = Subject name.

:barcode_tag = The unique barcode number assigned to the subject.

:rid_tag = The unique RFID number assigned to the subject.

:subject_type = Subject type name (Example: Bacteria or Sample).

:created_at = Date when the subject was created in the system.

:updated_at = Date when the subject was last updated.

:terminated = Flag if subject is terminated and no longer part of process/workflow.

:user = Username of a subject owner.

:permission = The permissions available to this subject.

:flow_states = Workflow states which the object has been in.

:created_by = The user that created the subject.

:updated_by = The user that last updated the subject.

:udfs = The number and names of all user-defined fields associated with this subject.

Section 2. List of API Functions

Following is a list of Methods along with the returned objects they call along with any required, query, or control parameters to use with them. See Section 3. List of API Examples for list of corresponding script examples. An explanation of each is as followed:

Returned Objects: LIMFINITY objects returned by API function.

Required Parameters: Necessary parameter or parameters for the method to run correctly. Without the required parameters an error message from the server should be expected. These will be written in the format required for the script along with placeholder text within the ‘ ’ marks.

Optional Query parameters: Optional parameters to control the results. These will be written in the format required for the script along with placeholder text within the ‘ ’ marks.

Optional Control parameters: Optional parameters to control the number and order of records in the output, and to implement paging of the results. These will be written in the format required for the script along with placeholder text within the ‘ ’ marks.

2.1 “audit”:

Retrieves a list of audit records of every change made within the system.

Returned Objects: AuditRecs

Required Parameters: None

Optional Query Parameters:

:date_flag=>’all/today/yesterday/week/month’ = Displays audit records made today, yesterday, this week, or this month.

:date_range=>’date_from,date_to’ = Displays audit records made within a specific date range(format: mm/dd/yyyy).

:subj_ids=>’value,value’ = Displays audits records of a specific range of subjects.

Optional Control Parameters:

:start=>’value’ = The item specific ID to start the list with.

:limit=>'value' = The total number of items to retrieve.

:sort=>'text' = Sort the displayed results by a specific field such as subject_type, subject_name, etc.

:dir=>'ASC/DESC' = Sort the displayed results in ascending or descending order.

2.2 “users”:

Retrieves a list of users within the system.

Returned Objects: Users

Required Parameters: None

Optional Query Parameters: None

Optional Control Parameters: None

2.3 “user_groups”:

Retrieves a list of user groups within the system.

Returned Objects: Groups

Required Parameters: None

Optional Query Parameters: None

Optional Control Parameters: None

2.4 “roles”:

Retrieves a list of roles within the system.

Returned Objects: Roles

Required Parameters: None

Optional Query Parameters: None

Optional Control Parameters: None

2.5 "userfields":

Retrieves a list of user-defined fields within the system.

Returned Objects: UserFields

Required Parameters: None

Optional Query Parameters:

:query=>'text' = Displays a list of user-defined fields containing a specific text string.

Optional Control Parameters:

:start=>'value' = The item specific ID to start the list with.

:limit=>'value' = The total number of results to retrieve.

:sort=>'text' = Sort the displayed results by a specific field such as subject_type, subject_name, etc.

:dir=>'ASC/DESC' = Sort the displayed results in ascending or descending order.

2.6 "subject_types":

Retrieves a list of subject types within the system.

Returned Objects: SubjectTypes

Required Parameters: None

Optional Query Parameters:

:query=>'text' = Displays a list of subject types containing a specific text string.

Optional Control Parameters:

:start=>'value' = The item specific ID to start the list with.

:limit=>'value' = The total number of results to retrieve.

:sort=>'text' = Sort the displayed results by a specific field such as subject_type, subject_name, etc.

:dir=>'ASC/DESC' = Sort the displayed results in ascending or descending order.

2.7 "subject_groups":

Retrieves a list of subject type groups within the system.

Returned Objects: SubjectTypeGroups

Required Parameters: None

Optional Query Parameters:

:query=>’text’ = Displays a list of subject type groups containing a specific text string.

Optional Control Parameters:

:start=>’value’ = The item specific ID to start the list with.

:limit=>'value' = The total number of results to retrieve.

:sort=>'text' = Sort the displayed results by a specific identifier such as subject_type, subject_name, etc.

:dir=>'ASC/DESC' = Sort the displayed results in ascending or descending order.

2.8 “subjects”:

Retrieves a list of subjects within the system.

Returned Objects: Subjects

Required Parameters:

:subject_type=>’value/text’ = Limits the search to all subjects of a specific type(can use subject type name or specific ID).

Optional Query Parameters:

:subject_ids=>’value,value’ = Limits subjects to a list of item specific IDs.

:subject_names=>’text,text’ = Limits subjects to a list of item specific names.

:query=>’text’ = Displays a list of subjects containing a specific text string.

Optional Control Parameters:

:start=>’value’ = The item specific ID to start the list with.

:limit=>'value' = The total number of results to retrieve.

:sort=>'text' = Sort the displayed results by a specific identifier such as subject_type, subject_name, etc.

:dir=>'ASC/DESC' = Sort the displayed results in ascending or descending order.

2.9 “delete_subject”:

Deletes a subject as specified by ID.

Returned Objects: None

Required Parameters:

:id=>’value’ = The specific ID of the subject to be deleted.

OR

:subject_type=>'value/text' = Limits the search to all subjects of a specific type(can use subject type name or specific ID).

:subject_name=>'text' = Limits the search to all subjects matching the specified name

Optional Query Parameters: None

Optional Control Parameters: None

2.10 "gen_token":

Creates an "auth_token" to be used instead of a user's password. Using the auth_token will omit the "API Session Created" and "API Session Removed" entries in the audit log.

Note: Generated Authorization Tokens are only valid for 10 minutes after they are either generated or last used. If an expired or invalid token is used, the following error message should be displayed: "Invalid Token".

Returned Objects: Auth_token

Required Parameters: None

Optional Query Parameters: None

Optional Control Parameters: None

2.11 "subject_details":

Deletes a subject as specified by ID.

Returned Objects: A list of values corresponding to a specific subject

Required Parameters:

:id=>'value' = The specific ID of the subject to be displayed.

:barcode_tag=>'value' = The specific barcode tag of the subject to be displayed.

:rfid_tag =>value = The specific RFID tag of the subject to be displayed.

Optional Query Parameters: None

Optional Control Parameters: None

2.12 "run_script":

Allows the running of a helper script by name and the passing of arbitrary JSON-formatted parameters to that script.

Returned Objects: Helper script-returned serialized JSON

Required Parameters:

:name=>'helper_script_name' = The name of the helper script to be run.

:data=>'' = Arbitrary data to be passed to the script represented as a JSON object(s) of a string with JSON-encoded object(s). This data will be accessible in the script as data.

*The script will also have access to all HTTP request parameters specified in the request using **params[:parameter_name]** syntax

**This function differs from all other API functions by how it should be called. Instead of passing "run_script" as a value for the "method" parameter, is requires a special URL in the following form:

http://{LIMS_ADDR}:{LIMS_PORT}/api/run_script

Example: http://demo.lims247.com/api/run_Script

Optional Query Parameters: None

Optional Control Parameters: None

2.13 "search_subjects":

Retrieves a subject or list of specified subjects.

Returned Objects: Subjects

Required Parameters:

:search_mode=>'REGULAR/DEEP' = Specifies the type of search to be performed.

:subject_type=>'value/text' = Limits the search to all subjects of a specific type(can use subject type name or specific ID, Example: 'Client').

:user_fields=>'text,text' = Comma delimited list of User-defined fields to retrieve for the searched subject(s)(Example: 'Address, Phone Number').

:fields=>'text' = fields to search within the subject(Example: 'Subject Name')

:conditions=>'text' = The conditions that will be applied to the search subject.

:values=>'value/text' = Specific strings within the subject to be searched for.

Optional Query Parameters: None

Optional Control Parameters:

:start=>'value' = The item specific ID to start the list with.

:limit=>'value' = The total number of results to retrieve.

:sort=>'text' = Sort the displayed results by a specific identifier such as subject_type, subject_name, etc.

:dir=>'ASC/DESC' = Sort the displayed results in ascending or descending order.

2.14.1 "import_subjects via CSV":

Imports a subject or subjects along with specified fields from a CSV file.

Returned Objects: Subjects

Required Parameters:

:file=>'.CSV file' = File to import(Must be CSV format).

:subject_type=>'value/text" = The specific type of the subject or subjects to be imported.

Optional Query Parameters: None

Optional Control Parameters: None

2.14.2 "import_subjects via JSON":

Imports a subject or subjects along with specified fields specified by subjects in JSON format

Returned Objects: None

Required Parameters:

:json=>'' = String of LIMFINITY subjects in JSON format.

:subject_type=>'value/text" = The specific type of the subject or subjects to be imported.

Optional Query Parameters: None

Optional Control Parameters: None

2.15 "upload_file_udf":

Uploads a file containing user-defined fields to the system.

Returned Objects: None

Required Parameters:

:udf_name=>' text' = Name of the file to be uploaded.

:file=>'text' = The content of the file to be uploaded.

Optional Query Parameters: None

Optional Control Parameters: None

2.16 "get_pedigree_data":

Retrieves the pedigree data of one family within the system.

Returned Objects: Key/value pair pf pedigree node subject and its information.

Required Parameters:

:family_subject=>'value/text' = Family subject type(can use subject type name or specific ID).

:pedigree_prop=>'text' = Pedigree type user-defined field or its name(not required if there is only one user-defined field for this type

Optional Query Parameters:

Optional Control Parameters:

Returned Hash Values: The key is an individual ID of a pedigree node. The key is the numeric subject ID for the pedigree nodes with the assigned subject, or a string otherwise. The value is a hash with the following content(each pair is optional):

:Individual = Assigned subject.

:Mother = Subject if the mother node has an assigned subject; String ID otherwise.

:Father = Subject if the father node has an assigned subject; String ID otherwise.

:Gender = ‘Male’ or ‘Female’.

:Sampled, :MZTwin, :DZTwin, :Proband, :Deceased, :Consultand, :Carrier, :Affected = Same as in the standard PED format.

:terminated_pregnancy = “true” if the pregnancy was terminated.

:adopted_in = “true” if the individual was adopted.

:adopted_out = “true” if the individual was adopted out.

Section 3. List of API Examples

Following is a number of Example API scripts written in ruby. Each example references the corresponding API method in Section 2. List of API Functions.

RURO has created a dedicated Limfinity VM for the API training. This VM and the corresponding credentials are referenced in the API script examples.

URL: https://api.limfinity.net

User: api_user

Password: !api_user123

3.1 Example 1 Audit:

API code below will print the total number of Audit Records and list all related information.

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "audit"
  }
```

```
http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "AuditRecs"
    value.each do |types|
      puts "--------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

Optional query parameters listed below can be added after *:method=> "audit"*

```
:date_flag=>'all/today/yesterday/week/month' #allows to search for a specific date
:date_range=>'date from,date to' #allows to search for a specific date range
:subj_ids=>'value,value' #comma-separated Subject IDs to get audit records
```

Optional control parameters listed below can be added after *:method=> "audit"*

```
:start=>'value' #specifies what record to start listing from
:limit=>'value' #limit number of records to retrieve
:sort=>'text' #sort the records by a specific value
:dir=>'ASC/DESC' #sort the records in ascending or descending order
```

3.2 Example 2 User:

This API will print the total number of Users and list all related information.

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "users"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "Users"
    value.each do |types|
      puts "---------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

3.3 Example 3 User Groups:

This API will print the total number of User Groups and list all related information.

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "user_groups"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)
```

```
data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "Groups"
    value.each do |types|
      puts "---------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

3.4 Example 4 Roles:

Print the total number of Roles and list all related information.

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "roles "
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "Roles"
    value.each do |types|
      puts "---------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

3.5 Example 5 Userfields:

Print the total number of User Fields and list all related information.

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "userfields"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "UserFields"
    value.each do |types|
      puts "--------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

Optional query parameters listed below can be added after `:method=>"userfields"`

```
:query=>'text' #optional search string to filter the results
```

Optional control parameters listed below can be added after `:method=>"userfields "`

```
:start=>'value' #specifies what record to start listing from
```

```
:limit=>'value' #limit number of records to retrieve
```

```
:sort=>'value' #sort the records by a specific value

:dir=>'ASC/DESC' #sort the records in ascending or descending order
```

3.6 Example 6 Subject Types:

Print the total number of Subject Types and any relevant information

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "subject_types"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "SubjectTypes"
    value.each do |types|
      puts "--------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

Optional query parameters listed below can be added after *:method=>"subject_types"*

```
:query=>'text' #optional search string to filter the results
```

Optional control parameters listed below can be added after *:method=>"subject_types"*

```
:start=>'value' #specifies what record to start listing from
```

```
:limit=>'value' #limit number of records to retrieve

:sort=>'value' #sort the records by a specific value

:dir=>'ASC/DESC' #sort the records in ascending or descending order
```

3.7 Example 7 Subject Type Groups:

Print the total number of Subject Type Groups and any relevant information

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "subject_groups"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "SubjectTypeGroups"
    value.each do |types|
      puts "---------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

Optional query parameters listed below can be added after *:method=>"subject_groups"*

```
:query=>'text' #optional search string to filter the results
```

Optional control parameters listed below can be added after *:method=>"subject_groups"*

```
:start=>'value' #specifies what record to start listing from

:limit=>'value' #limit number of records to retrieve

:sort=>'value' #sort the records by a specific value

:dir=>'ASC/DESC' #sort the records in ascending or descending order
```

3.8 Example 8 Subjects:

Print the total number of Subjects and any relevant information

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "subjects",
  :subject_type=>'Patient'
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
total = data['Total']
puts total
data.each do |key,value|
  if key == "Subjects"
    value.each do |types|
      puts "--------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```

Optional query parameters listed below can be added after *:method=>"subjects"*

```
:subject_ids=>'value' #limit subjects to a list of IDs
```

```
:subject_names=>'text' #limit subjects to a list of names

:user_fields=>'text' #comma-
delimited list of User Defined fields to retrieve for subjects

:query=>'text' #optional search string to filter the results.
```

Optional control parameters listed below can be added after *:method=>"subjects"*

```
:start=>'value' #specifies what record to start listing from

:limit=>'value' #limit number of records to retrieve

:sort=>'value' #sort the records by a specific value

:dir=>'ASC/DESC' #sort the records in ascending or descending order
```

3.9 Example 9 Create Subject and Delete Subject:

Create ‘Patient’ subject using ‘import_subjects’ method

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
data = {
    "MRN": '0000000001', "Subject ID": '9877665-01', "Date of Birth": '09/18/1982',
"Gender": 'Male'
}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "import_subjects",
  :subject_type=>'Patient',
  :json => data.to_json
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
new_id = data['ids']
puts "New ID: #{new_id}"
```

Delete subject

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "delete_subject",
  :id=>'34'
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
puts "---------------"
```

Optional query parameters can be used instead of `:id=>'value'`

```
:subject_type=>'value' #subject type name, like Patient

:subject_name=>'text' #subject name
```

3.10 Example 10 Update Subject:

Update 'Patient' subject using 'import_subjects' method. In the example below, values are updated for 2 fields: 'MRN' and 'Subject ID'

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
data = {
      "UID": '4',
      "MRN": '254415455',
      "Subject ID": '82619-105
}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "import_subjects",
  :subject_type=>'Patient',
  :json => data.to_json
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
record_id = data['ids']
puts "Updated subject. ID: #{record_id}"
```

3.11 Example 11 Generate Authentication Token:

Generate and print an "auth_token" to be used instead of a user password.

```
require 'net/https'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "gen_token"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

token = response.body
puts token
```

3.12 Example 12 Subject Details:

Print all details of a Subject as specified by ID, Barcode, and RFID Tag

```
require 'net/https'
require 'json'
```

```
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "subject_details",
  :subject_id => "4"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
data.each do |key,value|
        puts (key.to_s + ": " + value.to_s)
end
```

Optional parameters that can be used instead of `:subject_id`

```
:barcode_tag=>'text' #subject Barcode value

:rfid_tag=>'text' #subject RFID Tag value
```

3.13 Example 13 Run Script:

Execute Limfinity run_script / helper script by specifying the name of the script and the data which should be passed to it.

In this example, the script creates both Patient, Sample, Sample Type, and other records. Note that 'run_script' must be added to the end of the URL string.

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api/run_script')

header = {'Content-Type'=> 'application/json'}
data = {
    "Patient_MRN": '0000000001', "Patient_Subject ID": '9877665-01', "Patient_Date of
Birth": '09/18/1982', "Patient_Gender": 'Male',
  "Sample_Sample Type": 'Sample_Serum', "Sample_Volume": '10', "Sample_Units": 'mL',
"Sample_Date Collected": '08/20/19', "Sample_Date Received": '08/26/19'
}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :name=> "demo_helper_script",
  :data => data.to_json
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
puts "Success: #{data['success']}"
puts "Status: #{data['status']}"
puts "Message: #{data['msg']}"
```

Below is ‘demo_helper_script’ used together with the above API call.

```
# demo_helper_script used with a run_script API call
data = params[:data]
patient = nil
sample = nil
unit = nil
sample_type = nil
data.each_pair {|key, value|
  subject_type = key.split('_').first
  next if !['Patient','Sample'].include?(subject_type)
  field_name = key.split('_').last
  if subject_type == 'Patient'
      mrn = value if field_name =='MRN'
      patient = find_subjects(query:search_query(from:'Patient') {|qb|
        qb.compare('MRN', :eq, mrn)
      }).first if mrn.present?
      patient = create_subject('Patient') unless patient
      patient.set_value(field_name, value) if value.present?
  elsif subject_type == 'Sample'
    if field_name == 'Units'
      unit = find_subject({subject_type: 'Unit', name: value}) ||
create_subject('Unit', name: value)
    elsif field_name == 'Sample Type'
      sample_type = find_subject({subject_type: 'Sample Type', name: value}) ||
create_subject('Sample Type', name: value)
    else
```

```
      sample = create_subject('Sample') unless sample
      sample.set_value(field_name, value) if value.present?
    end
  else
    nil
  end
  }
sample.set_value('Patient', patient) if patient
sample.set_value('Units', unit) if unit
sample.set_value('Sample Type', sample_type) if sample_type
```

3.14 Example 14 Search Subjects:

API will search for a Subject of a specific Subject Type using provided query

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}

params = {
    :username => "api_user",
    :password=> "!api_user123",
    :method=> "search_subjects",
    :subject_type => "Patient",
    :query => "\"Subject ID\" LIKE '82619%' AND \"Date of Birth\" BETWEEN ('1950-01-
01','2000-01-01')"
}
http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
puts "Data: #{data}"
```

Optional control parameters that can be added after `:method=> "search_subjects"`

```
:start=>'value' #specifies what record to start listing from

:limit=>'value': limit number of records to retrieve

:sort=>'value': sort the records by a specific value
```

```
:dir=>'ASC/DESC': sort the records in ascending or descending order
```

3.15.1 Example 15.1 Import Subjects via CSV file:

Import Patient subjects from data in a CSV file

```
require 'net/https'
require 'json'
require "net/http/post/multipart"

url = URI.parse('https://api.limfinity.net/api')
csv_file = File.open("./import_file.csv")

csv_file.file do |f|
  request = Net::HTTP::Post::Multipart.new url.path,
  :username => "api_user",
  :password=> "!api_user123",
  :file=> UploadIO.new(f, "text", csv_file.filename),
  :method=>'import_subjects',
  :subject_type=>'Patient'

  http = Net::HTTP.new(url.host, url.port)
  http.use_ssl = true
  response = http.request(request)

  data = JSON.load(response.body)
  raise data.inspect
  puts "Success: #{data['success']}"
  puts "Status: #{data['status']}"
  puts "Message: #{data['msg']}"
end
```

Example of CSV file

```
Name,Subject ID,Description,MRN,Gender,Date of Birth

Patient 826A,82619-101,Patient record created via import_subjects CSV API,254871001,Female,2/5/1962

Patient 826B,82619-102,Patient record created via import_subjects CSV API,254871002,Male,2/10/1962

Patient 826C,82619-103,Patient record created via import_subjects CSV API,254871003,Female,2/28/1962
```

3.15.2 Example 15.2 Import Subjects via JSON:

API will import Subject using JSON

```
require 'net/https'
require 'json'
url = URI.parse('https://api.limfinity.net/api')
header = {'Content-Type'=> 'application/json'}

params = {
    :username => "api_user",
    :password=> "!api_user123",
    :method=>'import_subjects',
    :subject_type=>'Patient',
    :json  => {
    "Description": "Patient record created via import_subjects JSON API",
    "Date of Birth": "02/05/1962",
    "Gender": "Female",
    "MRN": "878114501",
    "Subject ID": "82619-003"
    }
}
http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)

data = JSON.load(response.body)
puts "Success: #{data['success']}"
puts "Status: #{data['status']}"
puts "Message: #{data['msg']}"
```

3.16 Example 16 Upload File UDF:

Upload a file to a field called 'File' as specified subject type/subject name combination

```
require 'net/https'
require 'json'
require "net/http/post/multipart"

url = URI.parse('https://api.limfinity.net/api')

File.open("./csv_file.csv") do |csv|
  request = Net::HTTP::Post::Multipart.new url.path,
  :username => "api_user",
  :password=> "!api_user123",
  :method=>'upload_file_udf',
  :file=> UploadIO.new(csv, "text", csv.name),
  :subject_type=>'Patient',
  :subject_name=>"#{'Patient 1'}",
  :udf_name=>'File'

  http = Net::HTTP.new(url.host, url.port)
  http.use_ssl = true
  response = http.request(request)

  data = JSON.load(response.body)
  puts "Success: #{data['success']}"
  puts "Status: #{data['status']}"
  puts "Message: #{data['msg']}"
end
```

Optional query - replace :subject_type=>'Patient',:subject_name=>"#{'Patient 1'}" with :id=>"4"

3.17 Example 17 Get Pedigree Data:

```
require 'json'
require 'net/https'

url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :password=> "!api_user123",
  :method=> "get_pedigree_data",
  :subject_type=>'Patient',
  :subject_id=>'4'
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)
data = JSON.load(response.body)
data.each_pair do |key, value|
  if key.is_a?(Numeric) #has assigned patient
    patient = value[:Individual]
    father = value[:Father]
    if father
      father_id = father.is_a?(Subject) ? father.id : father
      father_data = data[father_id]
      puts father_data #output father data
    end
    mother = value[:Mother]
    if mother
      mother_id = mother.is_a?(Subject) ? mother.id : mother
      mother_data = data[mother_id]
      puts mother_data #output mother data
    end
  else #No assigned patient
    puts "No assigned patient"
  end
end
```

3.18 Example 18 Using Auth Token:

This example is not linked to a corresponding method. The purpose of this example is to demonstrate how to use an auth token returned by "Example 10 Gen Token.rb".

Note: Auth tokens are only valid for ten minutes after they have either been created or last used. If an expired or invalid token is used, the following error message should be displayed: "Invalid Token".

```
require 'json'
require 'net/https'

url = URI.parse('https://api.limfinity.net/api')

header = {'Content-Type'=> 'application/json'}
params = {
  :username => "api_user",
  :auth_token=>'94d95700-06bd-4b8f-a9c4-66fbcc09c9ef',
  :method=> "users"
  }

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Post.new(url.request_uri, header)
request.body = params.to_json
response = http.request(request)
data = JSON.load(response.body)
total = data['Total']
puts total
puts data
data.each do |key,value|
  if key == "Users"
    value.each do |types|
      puts "--------------"
      types.each do |k,v|
        puts (k.to_s + ": " + v.to_s)
      end
    end
  end
end
```