iLab Solutions API

Last revised March 29, 2016
Contact api-support@ilabsolutions.com for questions

Table of Contents

Introduction

This development guide is intended for groups who are collaborating with iLab on projects that will use the API to deliver additional functionality. A sample application that authenticates against the API and performs simple price retrieval and updates is available as a learning resource. Please contact iLab if you would like a copy of the sample application and to obtain your first sets of client ID and tokens.

The current version of the API is meant to support a number of workflows. Here is an example of what can be achieved. Green steps occur in iLab, blue steps could occur via the api.

1. The customer or core initiate a service request in iLab
2. The core reviews the request and provides a quote
3. The PI, Lab or Department administrators approve financials when required
4. The core begins to process the request and through the API downloads key files and data from custom forms that are required by equipment processing requests
5. Depending on experiments run, the core updates the quantity of service delivered in iLab through the API and adds any charges that were not included up front.
6. The core reviews the request, clicks ‘complete’ on the project in iLab and then creates a billing event with those charges at the end of the month

Future versions of the API will support scheduling time on equipment and updating custom forms and uploading files.

1 - Authentication

The iLab API uses an implementation of the **OpenAuth 2 specification,** as drafted in May 2012. Version 1 of the iLab API uses the Bearer Token variation. Future versions will also include MAC encoding of tokens.

1.1. Obtaining a client ID and token

Client ID

A client ID serves to uniquely identify a client application to your core's API.

Token

Once a client ID has been generated, tokens can be associated with the client ID that provide a designated level of access to data through the API.

iLab will eventually make client ID and token generation self service. For now, please contact your iLab implementation associate to obtain your first sets of keys.

1.2. Using your client ID and token to request data through the API

The access token must be included in the header of each request to the API:

```
Authorization: bearer %token% \r\n
```

Here is example of a hand rolled request using telnet:

```
$ telnet api.server.local 80
Trying 127.0.0.1...
Connected to api.server.local.
Escape character is '^]'.
GET /v1/cores.xml HTTP/1.1
HOST: api.server.local
Authorization: bearer
/QcUOlaRdW8rzYBVcLLB0bHPFMzMiJPkTbsO0QGUxTRtc04mq6Xc6XqFt9rNphPyzBYRmsaT
1GmtnaClQ/ZK1A==
```

Each request to the API server must contain the access token. This will ensure that client applications have the appropriate access levels. This token must be kept secret on the client side, as it can represent a potential for a man-in-the-middle attack. Note, the iLab server will reject any requests that are not using SSL, but if you attempt plain-text access the token could

theoretically captured by a third party.

2 - Making Requests of the API

A Restful API

The iLab API is RESTfull, meaning it is aware of the HTTP verbs GET, POST, PUT and DELETE. It will perform the relevant action for the verb on the resource specified by the URL. Illustrative code examples will be included throughout the documentation:

GET	retrieve a resource from the API
POST	create a new resource of the type specified
PUT	update the resource specified in the URL
DELETE	delete the resource specified in the URL

XML and JSON

When you create or update a resource, you must pass the new data in the body of your request using the same XML or JSON structure that was used to retrieve the resource. You should also provide the correct headers specifying which data format you are using. For example, if you are passing in XML, you should add this HTTP header:

```
Content-Type: application/xml
```

When you GET data, the response type will depend on the extension you use in your request. For example...

```
GET https://api.ilabsolutions.com/v1/cores.xml
```

...will retrieve an XML response, while...

```
GET https://api.ilabsolutions.com/v1/cores.json
```

...will retrieve a JSON response. If you do not specify an extension, by default the API will return JSON

```
GET https://api.ilabsolutions.com/v1/cores
```

For consistency and readability, the rest of the examples will be given in XML.

Actions

Often, resources list the most important actions that can be performed on them. For example, for a service offered by a core, you would see an 'actions' node:

```
<actions>
  <view-prices>
    <url>/v1/cores/5582/services/493769/prices.xml</url>
    <method>GET</method>
  </view-prices>
  <update>
    <url>/v1/cores/5582/services/493769.xml</url>
    <method>PUT</method>
  </update>
  <delete>
    <url>/v1/cores/5582/services/493769.xml</url>
    <method>DELETE</method>
  </delete>
</actions>
```

The actions node contains the urls of actions that can be performed on the parent resource. While iLab is experimenting with HATEOAS in this first version of the API, future versions of the API will most likely rely on this structure and require less formal documentation.

2.1 Response format

All responses from the iLab API come wrapped in the ilab-response node:

```
<ilab-response>
      ...
</ilab-response>
```

The response may contain a node with metadata relevant to the response, such as pagination:

```
<ilab-response>
      <ilab-metadata>
      …
      </ilab-metadata>
</ilab-response>
```

As was described above, the API will return JSON or XML depending on the extension passed.

2.2 Pagination - response metadata

Pagination information is provided in the ilab-metadata object. Information includes the current page, and total number of items.

```
<ilab-response>
      <ilab-metadata>
            <next-page nil="true"/>
            <offset type="integer">25</offset>
            <previous-page type="integer">1</previous-page>
            <total type="integer">48</total>
            <total-pages type="integer">1</total-pages>
      </ilab-metadata>
      <!-- the collection would be here →
      ...
</ilab-response>
```

Changing pages is performed using the parameter page in get request

GET http://api-url/v1/resource **?page=2**

3 - Resource Overview

3.1 Cores

Cores are the root resource of the API. From a core, you should be able to navigate all of the resources and actions available.

GET - List of cores to which you access... /cores

```
GET https://api.ilabsolutions.com/v1/cores.xml
```

The response will contain an XML (or JSON) object listing all cores to which you have access, with the current implementation displaying only one core. You will see the name of the core, it's settings and a list of further available actions. An example XML response might look like this (some nodes have been collapsed for brevity):

```
<cores type="array">
  <core>
    <name>Molecular Cytogenetics Core</name>
    <settings>...</settings>
    <homepage>
      https://my.ilabsolutions.com/sc/5528/molecular-cytogenetics-core
    </homepage>
    <actions>
      <list-services>
        <url>
          https://api.ilabsolutions.com/v1/cores/5528/services.xml
        </url>
```

```
      <action>GET</action>
    </list-services>
    <list-equipment>
      <url>
        https://api.ilabsolutions.com/v1/cores/5528/equipment.xml
      </url>
      <action>GET</action>
    </list-equipment>
  </actions>
  <price-types type="array"/>
  <map>...</map>
 </core>
</cores>
```

GET - Details of a specific core you have access to /cores/:id

To retrieve the details of a specific core, you can GET a specific core ID. In the following example, 5582 is the id of the core you are interested in:

```
GET https://api.ilabsolutions.com/v1/cores/5582.xml
```

3.2 Services

You can GET the services offered by any core with the following URL:

GET - List of services in a core /cores/:id/services

```
GET https://api.ilabsolutions.com/v1/cores/5582/services.xml
```

GET - Details of a specific service in a core /core/:id/services/:id

You can also GET the details of individual services by requesting a URL in this format:

```
GET https://api.ilabsolutions.com/v1/cores/5582/services/493769.xml
```

This will return a service resource in the following XML format (again some nodes have been collapsed for brevity):

```
<service>
  <description><p>They were on ice</p></description>
  <name>Fish melting</name>
  <prices type="array">
```

```
    <price>
      <id type="integer">65</id>
      <price type="float">6.0</price>
      <actions>...</actions>
      <price-type>
        <id type="integer">13</id>
        <name>External</name>
      </price-type>
      <unit>
        <abbreviation>ea</abbreviation>
        <description>each</description>
        <id type="integer">37</id>
      </unit>
    </price>
    <price>...</price>
  </prices>
  <actions>...</actions>
  <category>
    <id type="integer">1862</id>
    <name>Main</name>
  </category>
</service>
```

If you want to update a service, use the same URL and use the PUT HTTP action:

```
PUT https://api.ilabsolutions.com/v1/cores/5582/services/493769.xml
```

In the body of the request, include the same XML you received when you retrieved the resource, but modify any fields that you would like to change. For example, if you want to change the first price and the name of the service you would pass on:

```
<service>
  <description><p>They were on ice</p></description>
  <name>Elaborate Fish Melting</name>
  <prices type="array">
    <price>
      <id type="integer">65</id>
      <price type="float">8.0</price>
      <price-type>
        <id type="integer">13</id>
        <name>External</name>
      </price-type>
      <unit>
        <abbreviation>ea</abbreviation>
        <description>each</description>
        <id type="integer">37</id>
      </unit>
```

```
    </price>
  </prices>
</service>
```

Normally, you'd only pass along the attributes that you want to update. If you want to set an attribute to nil, you need to specify that in the XML/JSON explicitly, e.g.:

```
<category nil="true"/>
```

To delete a service, build a DELETE request with the same URL:

```
DELETE https://api.ilabsolutions.com/v1/cores/5582/services/493769.xml
```

Please issue DELETE commands with care - In this case, the service and all of it's associated prices will be archived and no longer available for ordering.

Observations: Notice how the prices node's type equals "array". This most often designates a nested resource. In the case above, a service has many prices. Most resources have an 'id' attribute or node - this is the unique identifier for the resource.

3.3 Prices

You can view and update all or individual prices for a given service.

```
GET https://api.ilabsolutions.com/v1/cores/5582/services/493801/prices.xml
```

You can also view an individual price:

```
GET https://api.ilabsolutions.com/v1/cores/5582/services/493801/prices/66.xml
```

...which will return...

```
<price>
  <id type="integer">66</id>
  <price type="float">2.0</price>
  <actions>
    <update>
      <url>/v1/cores/5582/services/493801/prices/66.xml</url>
      <method>PUT</method>
    </update>
    <delete>
      <url>/v1/cores/5582/services/493801/prices/66.xml</url>
      <method>DELETE</method>
    </delete>
  </actions>
```

```
  <price-type>
    <id type="integer">12</id>
    <name>Internal</name>
  </price-type>
  <unit>
    <abbreviation>ea</abbreviation>
    <description>each</description>
    <id type="integer">37</id>
  </unit>
</price>
```

As indicated by the action node, you can update a price with the following URL...

```
PUT https://api.ilabsolutions.com/v1/cores/5582/services/493801/prices/66.xml
```

and passing through similar XML..:

```
<price>
  <id type="integer">66</id>
  <price type="float">4.3</price>
  <price-type>
    <id type="integer">12</id>
    <name>Internal</name>
  </price-type>
  <unit>
    <abbreviation>ea</abbreviation>
    <description>each</description>
    <id type="integer">37</id>
  </unit>
</price>
```

You can delete a price by calling...

```
DELETE
https://api.ilabsolutions.com/v1/cores/5582/services/493801/prices/66.xml
```

If a DELETE request is successful, a `204 - NO CONTENT` is expected.

3.4 Service requests

Retrieving Requests

For any core you can GET the following URL (also listed in the core's actions) to list the service requests:

GET - List of service requests belonging to a core /cores/:id/service_requests

```
GET https://api.ilabsolutions.com/v1/cores/5582/service_requests.xml
```

GET - Details of a specific service request belonging to a core /core/:id/service_requests/:id

You can also view individual service requests by requesting a URL of the type:

```
GET https://api.ilabsolutions.com/v1/cores/5582/services/493769.xml
```

POST - Create service request /cores/:id/service_requests

```
POST https://api.ilabsolutions.com/v1/cores/5582/service_requests.xml
<service-request>
      <owner_email>existing_user@email.com</owner_email>
      <pi_email>existing_user@email.com</pi_email> //optional
</service_request>
```

This request will create a service request with the state completed, which can be used as a shell to add additional charges.

Validations

Some validation of the owner_email and pi_email fields. It returns the following error codes and messages:

Error code	Error message
404	Owner not found
404	Owner Not have access to core or Owner is not an Employee of Core
404	Owner is not a member of any Group
404	PI not found
404	PI is not present in owner's groups

PUT - Update specific service request

/cores/:id/service_requests/:service_request_id

```
PUT https://api.ilabsolutions.com/v1/cores/5582/service_requests/493769.xml
<service-request>
      //fields to update
</service_request>
```

Following fields are updatable:
name, description, state, completed_on, start_on, end_on, quote_expires_on, has_recurring, projected_cost, summary

Filtering for requests

You can easily find requests of a particular status by passing through additional filter/query parameters.
For example:

```
GET https://api.ilabsolutions.com/v1/cores/5582/service_requests.xml?q=sample name&has_recurring=1&to_date=2013-03-12T12:54Z&states=cancelled,disagreement
```

Here is a list of the available filters that can be used to retrieve service requests:

Filter name	Available values
has_recurring	0 or 1 (optional)
from_date	string ISO 8601 formatted in UTC (optional)
to_date	string ISO 8601 formatted in UTC (optional)
q	string for fulltext search (optional)
order	order field name
states	valid request states(comma separated list): cancelled, completed, core_disagreement, disagreement, draft, equipment_scheduling, financials_approved, financials_rejected, needs_financial_reapproval, processing, proposed, requested, researcher_in_agreement, service_center_in_agreement

3.5 Service Request Rows

For any core you can GET the following URL (also listed in the core's actions) to list out service request rows. These include charges, milestones and custom forms.

```
<service-request>
      <service-rows>
            <service-row>
                  <position> 1</position>
                  <type>charge</type>
                  <id>1</id>
                  <name>name</name>
                  <actions></actions>
            </service-row>
      </service-rows>
</service-request>
```

3.6 Custom Forms

Custom forms are often used to collect important information from customers required by the core to deliver projects or services. They are associated to a service request through a service row.

Custom forms consist of fields. This list is not exhaustive, but the primary attributes of fields are the following:

attribute	description	notes/options
name	the name or label of the field	visible to the customer
type	the type of field to be displayed in the custom form	custom forms can include many standard form elements, including radio buttons, pull-down menus etc...see list below
value	the value entered by the customer	
default	the default value of the field	the default must be set in the custom form template
required	whether or not the field is a required field	

iLab's custom forms support special types that are worth highlighting:

field type	description	notes

help	a field for displaying help to the customer in the custom form.	this is a display only field and contains no customer input.
charges	charge fields allow the core to include a list of services in a custom form from which the customer can select and enter quantities	the service ID corresponds to a service that has been modeled on the core.
file / file_no_upload / file_import	file field types allow the core to provide the customer with a space to download templates and upload files.	

GET - list of custom forms
/v1/cores/:core_id/service_requests/:request_id/custom_froms.xml

GET
https://api.ilabsolutions.com/v1/cores/123/service_requests/35234/custom_forms.xml

```
<custom-forms type=”array”>
      <custom-form>...</custom-form>
      <custom-form>...</custom-form>
      …
      <custom-form>...</custom-form>
</custom-forms>
```

Here is an example custom form resource:

```
<custom-form>
  <id type="integer">29385</id>
  <name>Cell Sorting CLONE</name>
  <note>
    <p><a
    href="https://content.ilabsolutions.com/wp-content/uploads/2011/10/Sample-Qu
    estionaire.doc" target="_blank">Sample Questionaire</a></p><p>Available for
    download should you be submitting this on behalf of a new protocol.</p>
  </note>
  <fields type="array">
    <field>
      <name>
        I agree that my samples do not contain any infectious or radioactive
        material. The facility will refuse to sort my samples should they be
        labeled in such a way.
      </name>
```

```
  <show-if/>
  <required type="boolean">false</required>
  <default/>
  <type>select</type>
  <value>Yes</value>
  <choices>,Yes,No</choices>
</field>
<field>
  <name>Upload Sample Questionnaire:</name>
  <show-if>Is this a new protocol:=Yes</show-if>
  <required type="boolean">false</required>
  <default/>
  <type>file</type>
</field>
<field>
  <name>Protocol #:</name>
  <show-if>Is this a new protocol:=No</show-if>
  <required type="boolean">false</required>
  <default/>
  <type>string</type>
  <value>number</value>
</field>
<field>
  <name>Sample Questionnaire for new protocol:</name>
  <show-if>Is this a new protocol:=Yes</show-if>
  <required type="boolean">false</required>
  <default/>
  <type>file_no_upload</type>
</field>
<field>
  <name>Experiment Information:</name>
  <show-if/>
  <required type="boolean">false</required>
  <default></default>
  <type>text_section</type>
  <value></value>
</field>
<field>
  <name>Fluorochromes:</name>
  <show-if/>
  <required type="boolean">false</required>
  <default/>
  <type>string</type>
  <value>PE</value>
</field>
<field>
    <name>Services:</name>
```

```
    <show-if/>
    <required type="boolean">true</required>
    <default/>
    <processed>true</processed>
    <type>charges</type>
    <value type="array">
      <value>217701</value>
      <value>221103</value>
    </value>
    <required-services type="array">
      <required-service>
        <id>217701</id>
        <name>Media Preparation</name>
        <url>url placeholder</url>
        <quantity>2</quantity>
      </required-service>
      <required-service>
        <id>221103</id>
        <name>Next Day Expedite (Per Block)</name>
        <url>url placeholder</url>
        <quantity>0</quantity>
      </required-service>
    </required-services>
 </field>
<field>
    <name>Help Forms</name>
    <show-if/>
    <required type="boolean">false</required>
    <default>Does this show?</default>
    <type>help</type>
    <value>Does this show?</value>
  </field>
<field>
    <name><br/></name>
    <show-if/>
    <required type="boolean">false</required>
    <default>
      To track the shipment go to <a href="http://www.fedex.com/us/"
      target=""_new"">FedEx</a> and Enter the tracking number provided.
    </default>
    <type>text_section</type>
    <value>
      To track the shipment go to <a href="http://www.fedex.com/us/"
      target=""_new"">FedEx</a> and Enter the tracking number provided.
    </value>
  </field>
```

```
  </fields>
</custom-form>
```

Updating custom forms is not supported in the current version of the API.

GET - attachment from custom form /attachments/:attachment_id

Customers often submit data to cores in the form of an attachment, most commonly a csv file or an excel file. The value provided in the field is the attachment id. So to get the attachment make a request to:

GET https://api.ilabsolutions.com/v1/attachments/1231

3.7 Milestones

Cores often use milestones to organize and track important stages in the service request lifecycle.

GET - list of milestones /v1/cores/:core_id/service_requests/:request_id/milestones.xml

GET https://api.ilabsolutions.com/v1/cores/123/service_requests/35234/custom_forms.xml

```
<milestones type=”array”>
      <milestone>...</milestone>
      ...
      <milestone>...</milestone>
</milestones>
```

A milestone has the following properties

```
<milestone>
  <completed-on type="datetime" nil="true"/>
  <description/>
  <id type="integer">21214</id>
  <name>Sample Received</name>
  <started-on type="datetime" nil="true"/>
```

```
</milestone>
```

PUT - update milestone
/v1/cores/:core_id/service_requests/:request_id/milestones/:milestone_id.xml

To update milestone just send PUT request to the according url with new milestone data as below

PUT
https://api.ilabsolutions.com/v1/cores/5582/service_requests/493801/milestones/66.xml

```
<milestone>
  <completed-on type="datetime"></completed-on>
  <description>New description</description>
  <name>Sample Received</name>
  <started-on type="datetime" nil="true"/>
</milestone>
```

One can update following fields: ***completed-on, description, name, started-on***

3.8 Charges on Service Requests

To see the list of charges associated with a service request, one needs to

GET - List of charges in a service request
/cores/:core_id/service_requests/:service_request_id/charges.xml

GET
https://api.ilabsolutions.com/v1/cores/5582/service_requests/123/charges.xml

This will return a list of all charges in the following format:

```
<charges type=”array”>
      <charge>
            <id>12938</id>
            <name>Fetal Bovine Serum, Certified, Heat Inactivated</name>
            <quantity>1.0</quantity>
            <status>approved</status>
            <billing-status>billed</billing-status>
            <price-id>12093</price-id>
            <service-id>129378</service-id>
      </charge>
      ...
```

```
</charges>
```

POST - Add charges to a service request /cores/:core_id/service_requests/:service_request_id/charges.xml

Charge rows can be added to a service request through the API. To successfully add a charge to a service request, you must be able to identify the service request, the service and you must also indicate the quantity of the service you would like to charge. The default payment information stored on the service request will be associated with the charge.

POST
https://api.ilabsolutions.com/v1/cores/5582/service_requests/123/charges.xml
with post data:

```
<charges>
      <charge>
            <quantity>1.5</quantity>
            <price-id>128398</price-id>
            <service-id>12947739</service-id>
      </charge>
      <charge>
            <quantity>1.0</quantity>
            <price-id>128398</price-id>
            <service-id>12947729</service-id>
      </charge>
</charges>
```

The above command would add two charges for the service indicated.

The following attributes can be updated on a charge: name(only if core setting “Allow managers to edit line item name” if set to TRUE), status, billing status, quantity. To update charges, issue the following command...

PUT - Update charge information charges to a service request /cores/:core_id/service_requests/:service_request_id/charges/:charge_id.xml

PUT
https://api.ilabsolutions.com/v1/cores/5582/service_requests/123/charges/123.xml

with the following request body

```
      <charge>
            <name>Fetal Bovine Serum, Certified, Heat Inactivated</name>
            <quantity>1.5</quantity>
```

```
        <billing-status>not billed</billing-status>
        <status>rejected</status>
    </charge>
```

By default, a billing status and status are automatically set when a charges is added to a request - these values depend on the status of the parent service request. The following are allowed billing and work statuses in case you need to update them:

Billing State	notes
cancelled	the billing status of any charge that have been cancelled and will not be billed for.
not_ready_to_bill	the billing status when charges are being processed by the core but are not ready to bill yet.
ready_to_bill	the status when a charges is ready to be included in a billing event. typically, cores bill once a month and when a new billing event is generated, by default all **ready_to_bill** events are included on the billing event.
not_billable	if a charges is not billable for some reason such as poor sample quality, it can be marked as **not_billable**
pro_bono	if there is an agreement to perform a particular piece of work pro-bono, the**pro_bono** billing status may be used
billing_initialized	billing for charges must occur in iLab by creating and processing billing events
billed	billing for charges must occur in iLab by creating and processing billing events
paid	billing for charges must occur in iLab by creating and processing billing events

Status	notes
proposed	when a request is in draft mode or has not yet been approved, the status is typically**proposed.**
financials_approved	the status applied to charges once a service request has been approved by an authorized user in iLab. when a service request has been approved, new charges automatically receive a status of **financials_approved**

processing	if desired, a core can indicate that they are processing a particular charge. this is often used when a charge represents a specific service that will be performed.
completed	when the work status is updated to completed, it is assumed that the charge is ready to bill. in the ilab interface, when a core marks a charges as completed, the status automatically updates to **ready_to_bill**, unless the billing status is pro_bono, not_billable, cancelled or any of the post billing states in red.
cancelled	

4 - Error handling

There are several types of errors that the iLab API will return in case there is something wrong with your request or the API server:

4.1. 422 Bad Request

This may be the most frequent error you get from the API. It usually means that validation has failed on the data you sent for an update or create request. This could be because of the formatting on date and time fields, required fields missing, or some other similar problem with the input data. The body of the response should contain a description of the error which should help you address its causes.

4.2. 401 Unauthorized

You will get this error when either the access token you have passed is invalid, non-existent, or it is not authorized to perform the action you are trying to perform. Some access tokens have expiration dates and need to be renewed on a regular basis.

4.3. 404 Not Found

You may get this error if you are trying to access a resource that doesn't exist or if the URL you are trying to use to access the resource is invalid. Please check your URL logic and have in mind the resource may have been deleted via the main iLab application or by another client application.

4.4. 500 Internal Server Error

Hopefully you should not get this error very frequently, it probably means there is some kind of misconfiguration on the server side, which may or may not be related to the actual data or action you are trying to perform. The iLab team will be notified of these errors and will try and fix them as soon as possible, but feel free to tell us about what you were trying to do and where it went wrong.