iLab Solutions API

Last revised February 15, 2023
Contact api-support.pdl-ilab@agilent.com for questions

Table of Contents

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

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

Validations

PUT - Update specific service request /cores/:id/service_requests/:service_request_id

Filtering for requests

3.5 Service Request Rows

3.6 Custom Forms

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

GET - attachment from custom form /attachments/:attachment_id

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

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

3.8 Charges on Service Requests

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

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

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

3.9 Equipment list

GET - List of equipment /cores/:core_id/equipment.xml

3.10 Attachments on Service Requests

GET - Download attachment from a service request /v1/attachments/:attachment_id

POST - Add attachment to a service request /v1/attachments

DELETE - Delete an attachment from a service request v1/attachments/:attachment_id

4 - Error handling

4.1. 422 Bad Request

4.2. 400 Bad Request

4.3. 401 Unauthorized

4.4. 404 Not Found

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.

Current API Instances

Please contact api-support.pdl-ilab@agilent.com for a list of the current API instances

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.

1.1. Obtaining a client ID and token

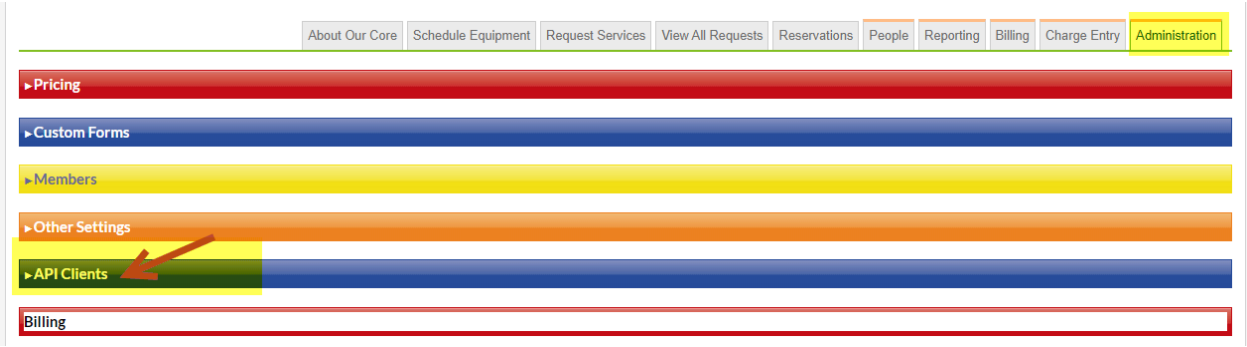
Client ID

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

Token

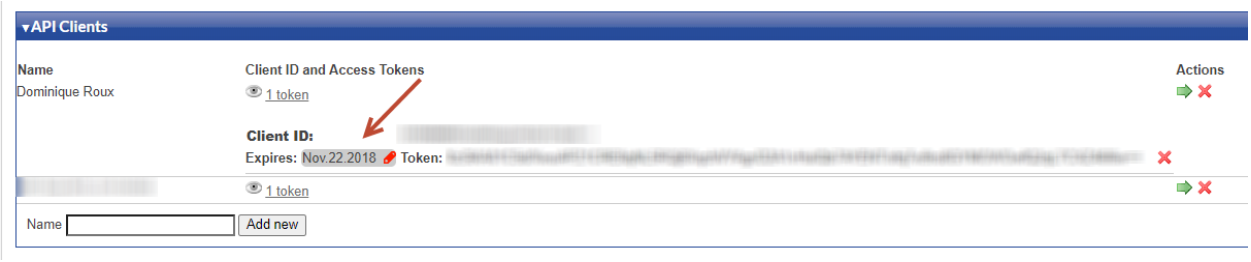
Once a client ID has been generated, tokens can be associated with the client ID that provide a designated level of access to data through the API.

iLab has now made core-level client ID and token generation self-service if your institution has enabled API access for their core facilities. If your institution has enabled API access, the *API Clients* section will be visible within a core’s Administration tab to core administrators:



Please note that tokens, by default, expire one year from the date of initial generation. If you would like to set a longer expiration period for your token, take the following steps:

Go to the *API Clients* section within your core Administration tab. Click on the link that says "# token" to expand the details of the particular token that has expired. Place your cursor over the expiration date and a red pencil will appear. Click on the date and a calendar pop-up will appear that will allow you to set a new expiration date:



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
```

One can use Postman chrome extension to run requests manually
https://chrome.google.com/webstore/category/extensions?hl=en

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 RESTful, 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://{domain}.com/v1/cores.xml
```

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

```
GET https://{domain}.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://{domain}.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/1234/services/493769/prices.xml</url>
    <method>GET</method>
  </view-prices>
  <update>
    <url>/v1/cores/1234/services/493769.xml</url>
    <method>PUT</method>
  </update>
  <delete>
    <url>/v1/cores/1234/services/493769.xml</url>
    <method>DELETE</method>
```

```
  </delete>
</actions>
```

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 the 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://{domain}.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/1234/molecular-cytogenetics-core
    </homepage>
    <actions>
      <list-services>
        <url>
          https://{domain}.com/v1/cores/1234/services.xml
        </url>
        <action>GET</action>
      </list-services>
      <list-equipment>
        <url>
          https://{domain}.com/v1/cores/1234/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, 1234 is the id of the core you are interested in:

```
GET https://{domain}.com/v1/cores/1234.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://{domain}.com/v1/cores/1234/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://{domain}.com/v1/cores/1234/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>
      <public_visibility=”integer”>0</public_visibility>
      <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://{domain}.com/v1/cores/1234/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"/>
```

Public Visibility setting controls if service is visible on the core's landing page and searchable on the institution landing page. Allowed values of public_visibility allowed include:

- 0 not visible on landing page
- 1 visible on landing page

3.3 Prices

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

```
GET https://{domain}.com/v1/cores/1234/services/493801/prices.xml
```

You can also view an individual price:

```
GET https://{domain}.com/v1/cores/1234/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/1234/services/493801/prices/66.xml</url>
      <method>PUT</method>
    </update>
    <delete>
      <url>/v1/cores/1234/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://{domain}.com/v1/cores/1234/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://{domain}.com/v1/cores/1234/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://{domain}.com/v1/cores/1234/service_requests.xml
```

By default this will return data from the last two years. You can pass the following parameters to modify the returned data scope:

from_date (datetime): Only data after this will be included. Default is 2 years ago.
per_page (integer): The number of requests returned per page. Default is 30.

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://{domain}.com/v1/cores/1234/service_requests/493769.xml
```

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

```
POST https://{domain}.com/v1/cores/1234/service_requests.xml
<service_request>
      <owner_email>existing_user@email.com</owner_email>
      <pi_email>existing_user@email.com</pi_email> //optional
      <name>Optional Request name</name> //optional
      <state>processing</state> //optional, default is 'completed'.
</service_request>
```

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

List of parameters:

- owner_email - Email of the user in iLab. user will become an owner of the request.
- pi_email – email of the PI of the group, to which to assign th request. Owner should be user of the group. Used in case user has multiple groups.
- name – sets Service request name in the system. Default is autogenerated, based on the settings.
- state – override the default state (“completed”). List of allowed states: proposed, needs_financial_reapproval, processing, completed

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://{domain}.com/v1/cores/1234/service_requests/493769.xml
<service_request>
      //fields to update
</service_request>
```

The following fields can be updated:
name, description, state, completed_on, start_on, end_on, quote_expires_on, has_recurring, projected_cost, summary

Please find the transitions between states described here.

List of states one can set via API:

- proposed
- needs_financial_reapproval
- processing
- completed

Filtering for requests

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

```
GET https://{domain}.com/v1/cores/1234/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	Default values
has_recurring	0 or 1 (optional)	Both 0 and 1
from_date	string ISO 8601 formatted in UTC (optional)	Time.now - 2.years
to_date	string ISO 8601 formatted in UTC (optional)	Time.now + 1.day
q	string for fulltext search (optional)	Not applied
name	string for exact match search by name (optional)	Not applied
order	order field name	created_at
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	All states

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_forms.xml

GET https://{domain}.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-Questionaire.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://{domain}.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://{domain}.com/v1/cores/123/service_requests/35234/milestones.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://{domain}.com/v1/cores/1234/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 the 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://{domain}.com/v1/cores/1234/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>
            <note>Additional note</note>
      </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://{domain}.com/v1/cores/1234/service_requests/123/charges.xml
with post data:

```
<charges>
      <charge>
            <quantity>1.5</quantity>
            <price-id>128398</price-id>
            <service-id>12947739</service-id>
            <note>Additional note (Optional)</note>
      </charge>
      <charge>
            <quantity>1.0</quantity>
            <price-id>128398</price-id>
            <service-id>12947729</service-id>
            <note>Additional note 2 (Optional)</note>
      </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, note. 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://{domain}.com/v1/cores/1234/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>
            <note>Updated Note(Optional)</note>
      </charge>
```

By default, a billing status and status are automatically set when a charge 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	

3.9 Equipment list

List equipment registered in the core

GET - List of equipment /cores/:core_id/equipment.xml

GET https://{domain}.com/v1/cores/1234/equipment.xml

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

```
<?xml version="1.0" encoding="UTF-8"?>
<ilab-response>
  <equipment type="array">
    <equipment>
      <id type="integer">1</id>
      <name>Core Equipment #1</name>
      <pubilc_visibiity>1</public_visibility>
      <description>Some description</description>
      <url>https://my.ilabsolutions/equipment/1</url>
    </equipment>
    ...
  </equipment>
</ilab-response>
```

Each equipment node describeds the following:

- id – iLab internal ID of the equipment
- name – name of the equipment in iLab
- description – description of the equipment in iLab
- url – URL of the equipment in the iLab Web Interface. Following this URL in browser will open a page of the equipment in iLab.
- public_visibility – describes visibility on landing page. Can be 0 or 1 or 2
    - 0 – Not visible

- 1 – Visible
- 2 – Visible and also calendars are accessible publicly.

3.10 Attachments on Service Requests

Permissions for attachments are based on the users permissions for the core the service requests belong to. Read access to the core is required to download attachments and update access is required to add/remove attachments from a service request.

GET - attachment from service request /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://{domain}.com/v1/attachments/1231

POST - add an attachment to a service request /attachments

To add an attachment to a service request. Must include the following params

```
object_class: "ServiceItem" This is case sensitive and will always be the same
id: iLab internal ID of the service request
```

POST https://{domain}.com/v1/attachments?object_class=ServiceItem&id=163

The attachment must be added to the body along with an optional metadata tag to name it as follows:

```
attachment[uploaded_data]: file
attachment[name]: "Desired Name" (optional)
```

This will return the service request that the attachment is added to in the same format as a GET request for that single service request (which includes all of the attachments associated with that request)

The filename cannot contain a null byte string in it ("\u0000")
The following file extensions are blacklisted: .bat, .exe, .tar
The following content-types are blacklisted:

```
application/bat
application/dos-exe
application/exe
application/gnutar
```

```
application/msdos-windows
application/x-bat
application/x-exe
application/x-ms-dos-executable
application/x-msdos-program
application/x-msdownload
application/x-tar
application/x-winexe
multipart/x-tar
vms/exe
```

DELETE - delete an attachment from a service request /attachments/:id

This will delete an attachment based on its attachment_id

DELETE https://{domain}.com/v1/attachments/1231

This will return the service request that the attachment is added to in the same format as a GET request for that single service request (which includes all of the attachments associated with that request)

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. 400 Bad Request

This error indicates that the server (remote computer) is unable (or refuses) to process the request sent by the client, due to an issue that is perceived by the server to be a client problem (for example, malformed request syntax, invalid request message framing, deceptive request

routing, and incomplete data). Example: A 400 response code will now be returned when a ServiceID or PriceID is incorrect (or expired, in the case of PriceIDs).

4.3. 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. By default, access tokens expire 1 year from the day of generation and may need to be renewed on an annual basis. Alternatively, please contact your iLab representative if you would initially like your token expiration date to be set for a period of > 1 year.

4.4. 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.

4.5. 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.

Changelog

2017-03-14

1. Added **name** filter for service_request search. New filter is exact match filter.
2. Fix for inconsistent format of data in **actions** section.
    a. All the urls there now are full urls.
    b. HTTP verb is provided in **action** and **method** attributes.
3. PriceType resource collection render format is now consistent. Elements in the collection are not marked with a key now.
   { "price_types": [{"price_type": {..}, {"price_type": {..}] } - WAS
   { "price_types": [{..}, {..}] } - NOW

4. Links to non-functional resources removed from the **core** resource actions list:
    a. List equipment:
    "list_equipment": {"url": "https://example/v1/cores/2917/equipment.json", … }
    b. List categories:
    "list_categories": {"url": "https://example/v1/cores/2917/categories.json", … }
5. Core settings rendering has been fixed to show all settings available to the user.

2018-02-14

Removed section about service delete action - which is not available, thus misleading.

2018-03-10

- Default payment information is rendered for Service Request:

```
<service_request>
  ...
  <cost_allocations>
    ...
  </cost_allocations>
  ...
</service_request>
```

- Changing requests status from processing to completed will automatically complete the charges on the request and mark them as ready_to_bill where applicable.

2018-05-28

- Service request is now providing the <service_name> which corresponds to the service/service project template name that was requested.

2018-07-13

Sections representing people(owner, principal_investigator) will now have **employee_id** attribute exposed. Employee_id is the field which iLab software receives from a customer ERP or IdP as a unique user identifier:

XML

```
        <owner>
            <id type="integer">26384</id>
            <name>Pavel Shegai</name>
            <first-name>Pavel</first-name>
            <last-name>Shegai</last-name>
            <email>pavel.shegai@ilabx.com</email>
            <phone>123123131</phone>
```

```
            <employee-id>pavel.shegai</employee-id>
        </owner>
```

or JSON

```
    "owner": {
                "id": 26384,
                "name": "Pavel Shegai",
                "first_name": "Pavel",
                "last_name": "Shegai",
                "email": "pavel.shegai@ilabx.com",
                "phone": "123123131",
                "employee_id": "pavel.shegai",
            },
```

2019-05-22

Bugfix: services endpoint has been rendering services deleted from the UI. We use “soft-delete” mechanism which was not respected by API. This has been fixed. Now services removed from the UI will not be available through the API.

2019-12-10

Feature: added a resource to list equipment in the core.

Feature: ability to override name and default status of the Service request

2020-04-30

Feature: Exposed public_visibility attribute for equipment and services

Bugfix: Consistent pagination metadata for paginated collections: Charges, Equipment, Services, Milestones, Service Requests

Bugfix: Deleted and Draft Equipment is not exposed via API.

2020-09-12

Feature: Charge note is now available for create, read and update as a part of the Charge. Note field is optional

2021-12-31

Bugfix: Access tokens are now removed upon deletion of the client/token. Deleted tokens could previously still be used to authenticate against the API. This has been fixed.

2022-01-08

Bugfix: API tokens with read-only access were able to create/update/destroy service requests/charges/prices. This has been fixed.

2022-03-12

Enhancement: Added new 400 response code to address the following scenarios: a 400 Bad Request error indicates that the server (remote computer) is unable (or refuses) to process the request sent by the client due to an issue that is perceived by the server to be a client problem (for example, malformed request syntax, invalid request message framing, deceptive request routing, and incomplete data). Example: A 400 response code will now be returned when a ServiceID or PriceID is incorrect (or expired, in the case of PriceIDs).

2023-01-07

Enhancement: Added new endpoints for creating and deleting attachments for service requests. Also added the filename as part of the response for service requests under the 'attachments' header

2023-01-29

Bugfix: assigned_to value on service requests objects will now return an array of names of people that the service has been assigned to. It previously returned null.