

LabCAS Data Upload Documentation

Laboratory Catalog and Archive Service (LabCAS) is designed to support the processing, capture, curation and sharing of data before publications. LabCAS is a secure system that requires an EDRN username and password. The system will allow users to securely share data with collaborators prior to publication and push to a public repository upon approval.

Step-by-step guide to deposit data:

1. **Login** to LabCAS using your EDRN account at <https://labcas-dev.jpl.nasa.gov/manage/datasets>. You will see a *My Data* tab and a *Public Data* tab. The *My Data* tab lists all datasets that you have permission to view and download.
If you do not have an EDRN account please contact the EDRN Informatics Center at edrn-ic@jpl.nasa.gov.
2. Scroll to the bottom of the page and click on the **Upload** button.
3. This page lists different types of data that LabCAS supports such as a standard EDRN dataset or specific data pipelines that have been created. Click on the appropriate type of data you would like to upload. For most users this will be the **Standard LabCAS Dataset**.
4. Enter the **Metadata** for the collection of files you are uploading.
 - a. **Dataset Identifier** - Enter a unique identifier for the dataset that will be used by the LabCAS system. [ShortProtocolName] [ShortSiteName][SitePILastName] or [DatasetName][ShortSiteName][SitePILastName]
 - b. **Protocol Name** - Start typing either the Protocol Name or Protocol ID associated with this data and select the protocol from the list.
If you do not have an EDRN protocol created for this data, please contact the EDRN Informatics Center at edrn-ic@jpl.nasa.gov. You cannot upload data without a protocol.
 - c. **Dataset Name** - Enter a descriptive name for the data collection you are uploading.
 - d. **Lead PI** - Start typing the name of the PI and select name from the list.
 - e. **Data Custodian** - Enter the full name of the data custodian. First and last name.
 - f. **Data Custodian Email** - Enter the email address of the data custodian.
 - g. **Collaborative Group** - Select the appropriate EDRN Organ Collaborative Group or N/A if that does not apply.
 - h. **Share data with...** - Select what groups can access your data. By default the system selects all groups that are associated with your login. Groups are defined in EDRN by the PI last name and institution name. You can click the "x" to the right of the name to remove access for a group. To add additional groups, click **Add Group** and start typing the PI last name and select them from the list.
5. Click **Submit**.
6. Click **Browse** or **Drag** the files to upload into the light gray box.
7. Click **Start Uploading** once you see all of the correct files listed in the Staged for upload list.
8. To add additional files you can **Browse** or **Drag** them into the light gray box and click **Start Uploading** again. Repeat this process until all files have been uploaded.
9. Click **Finish** when you are done uploading files.
10. You are returned to the *My Data* tab. You may need to **refresh** your browser to see your new dataset in your list. For larger data the upload may take a few minutes.

Related articles

- [LabCAS Data Upload Documentation](#)

LabCAS APIs

LabCAS includes the ability to query and download data programmatically, i.e. using scripts and programs that interact with REST-based services deployed as part of the LabCAS infrastructure at JPL or in the AWS cloud. All operations are subject to authentication and authorization: users have access to the same data and metadata that is visible to them through the LabCAS front-end web portal.

A typical API-based workflow to download a large number of files involves 3 steps:

1. Query LabCAS to identify data of interest
2. Obtain a list of files to download, by their URL
3. Use any HTTP client to download the data specified by those URLs

The user interacts with the LabCAS services through the HTTP client of their choice - a python script, a standard program such as **CURL** or **WGET**, etc. The client of choice must be able to support SSL, as the user credentials or token must be transmitted securely as part of the client request.

The following holds true for all API endpoints:

- On the client side, HTTP GET parameter values **must** be properly URL-encoded
- On the client side, HTTP POST payloads (in JSON or XML format) **should not** be URL-encoded
- On the server side, all user input (URL-decoded HTTP GET parameter values and HTTP POST payloads) is checked for unsafe characters (">", "<", "%", "\$", "\"): if found, the request will be rejected

In what follows, we use CURL as example client, and we interact with the MCL LabCAS server at "mcl-labcas.jpl.nasa.gov". The same functionality is supported by the EDRN LabCAS server at "edrn-labcas.jpl.nasa.gov", or the prototype LabCAS servers for MCL/EDRN/CIB deployed on AWS.

- [Authentication and Authorization](#)
- [Query API](#)
- [Listing API](#)
- [Download API](#)
- [Bulk Data Download Example](#)
- [User Data API](#)

Authentication and Authorization

All LabCAS API endpoints (query, list, download, user data) are subject to access control: the HTTP request must include the user credentials, so that the system can: a) authenticate the user, and b) verify that the user is authorized to access the requested data. The LabCAS APIs support 2 possible authentication schemes:

- Standard username and password, transmitted in the HTTPS "Authorization: Basic" header
- Jason Web Tokens, transmitted in the HTTPS "Authorization: Bearer" header

Authentication via username and password

Users can authenticate to the LabCAS back-end services by providing their username and password through the standard HTTP(S) Basic Authentication protocol. Specifically, username and password are URL encoded, base64 encoded and transmitted as part of the "Authorization: Basic" header. For example:

- Authorization: **Basic aHR0cHdhbGNoOmY=**

When using CURL, username and password can be specified in two ways (replace "mylogin" and "mypassword" with your actual LabCAS username and password):

- On the command line through the **--user** option:
 - Example: `curl --user mylogin:mypwd 'http://mcl-labcas.jpl.nasa.gov/data-access-api/...`
 - This option is not recommended because username and password can be seen by any user on the system who decides to display all processes.
- More securely, using the **--netrc** option, which assumes that username and password are written in a file called `~/.netrc` in the user home directory (`_netrc` on Windows) which should be readable only by that user.
 - Example: `curl --netrc 'http://mcl-labcas.jpl.nasa.gov/data-access-api/....`
 - The file `~/.netrc` can contain username/password for several servers, in the format below:

```
$ cat ~/.netrc
machine mcl-labcas.jpl.nasa.gov
login mylogin
password mypassword
machine edrn-labcas.jpl.nasa.gov
login mylogin2
password mypassword2
```

Authentication via Json Web Tokens (JWT)

JWT is an industry standard for transmitting signed token information. The LabCAS backend services will accept JWT tokens generated and signed by a trusted identity provider such as the LabCAS front-end. The two parties share a secret key which is used to sign the generated token (front-end), and validate the token received (back-end). There are libraries in many programming languages that can be used to facilitate the process of creating and validate JWTs (see <https://jwt.io/>).

In LabCAS JWTs are used to carry either user or product information, depending on the end point. The payload of a LabCAS JWT always contains the following fields:

- iss (aka issuer): set to "LabCAS"
- aud (aka audience): set to "LabCAS"
- exp (aka expiration): time after which the token should NOT be considered valid (in seconds part the Unix Epoch)
- iat (aka issued at): time when token was issued (in seconds part the Unix Epoch)
- nbf (aka not before): time before which the token should NOT be considered valid (in seconds part the Unix Epoch)
- sub (aka subject): this can be either the user Distinguished Name (when carrying user identity), or the product download id (when carrying product information)
 - Example: "sub": "uid=lcinquini,ou=users,o=MCL"
 - Example: "sub": "bac2152f-f013-450a-b1ae-89e24720dbdf"

The time fields are automatically used to validate the token (in addition to the token signature). The subject field is used to:

- By the Query and List API, retrieve a trusted user identity, which is then used to query LDAP for all the access control groups the user is part of, in order to sub-select the metadata records that the user is allowed to view
- By the Download API, to authorize download of a specific product identified by its id (an no other product).

LabCAS JWTs expire after 60 seconds, so even if stolen, they would be of limited use.

Valid LabCAS tokens can be generated by trusted applications (i.e. applications that share the LabCAS secret key for tokens), or can be retrieved from LabCAS itself by providing valid credentials (username and password). The following request retrieves a valid token directly from LabCAS:

- `TOKEN=`curl -Lkv --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/auth'`

Bulk Data Download Example

This example demonstrates how to download all files belonging to a LabCAS collection. Specifically, we will download all files that are part of the EDRN collection "Automated Quantitative Measures of Breast Density Data". These instructions assume you are working on a Unix-like system (including MacOSX), although they can be applied with small changes to Windows-based systems as well.

To gain more in-depth knowledge, please consult the full documentation for the [LabCAS APIs](#).

Pre-Requisites

- You must have a valid EDRN username and password
- You must have written your EDRN username and password in the home directory file `~/.netrc` in the following format:

```
prompt$ cat ~/.netrc
machine edrn-labcas.jpl.nasa.gov
login <your EDRN username>
password <your EDRN password>
machine ... some other host ...
login ...the other host username...
password ..the other host password...
...
```

Identify the Data

Visit the EDRN web site <https://edrn-labcas.jpl.nasa.gov/> to identify the data of interest (you might need to log-in). In particular, you can browse to the top-level collection page: https://edrn-labcas.jpl.nasa.gov/ui/c/Automated_Quantitative_Measures_of_Breast_Density_Data and write down the collection identifier, which in this case is: "Automated_Quantitative_Measures_of_Breast_Density_Data".

Alternatively, you could use the LabCAS Query API to search for the desired collection via some search constraints, and return the id as part of the result.

- syntax: `https://<hostname>/data-access-api/collections/select?q=<your query>`

For example, the following command searches for the collection with that specific CollectionName (note that the HTTP parameter value is URL-encoded):

- `curl --netrc 'https://edrn-labcas.jpl.nasa.gov/data-access-api/collections/select?q=CollectionName:%22Automated%20Quantitative%20Measures%20of%20Breast%20Density%20Data%22&wt=json&indent=true'`

List All the Files

Use the "list" endpoint to list the URLs for all files that belong to a specific collection, specified by its identifier.

- syntax: `https://<hostname>/data-access-api/collections/list?q=id:<the collection id>`

For example, use the following command to create a listing of URLs for all files that belong to that collection:

- `curl -Lkv --netrc 'https://edrn-labcas.jpl.nasa.gov/data-access-api/collections/list?q=id:Automated_Quantitative_Measures_of_Breast_Density_Data&wt=json&indent=true' > urls.txt`

The file `urls.txt` will contain all the download URLs.

Download All Files

The following command will invoke the cURL client to download each single file referenced in `urls.txt` (9649 files for this specific collection):

- `xargs -n 1 curl -O -g --netrc --verbose --location --remote-name --remote-header-name --cookie-jar /tmp/jar < urls.txt`

Download API

The Download API allows to download one file at a time, using Basic authentication or Jason Web Tokens. A file is identified by its unique "id". The general syntax of a query request is:

- `https://<server host name>/data-access-api/download?id=...`

where:

- `<id>` = file unique identifier

Examples with Basic Authentication:

- `curl --netrc -O -J -L -v 'https://mcl-labcas.jpl.nasa.gov/data-access-api/download?id=VUMC_Lung_Adenocarcinoma_CyTOF_data/CyTOF_ADC_prelim/preprocessed/031418_11938_I_pp.fcs'`

Example with Jason Web Token:

- `TOKEN = `curl -Lkv --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/auth``
- `curl -H "Authorization: Bearer $TOKEN" -O -J -L 'https://mcl-labcas.jpl.nasa.gov/data-access-api/download?id=Boston_University_Lung_Tumor-Sequencing.FFPE_Lung_Tumor-Sequencing.31551_S2_L001_R1_001.fastq.gz'`

Note that any HTTP client -of any kind- can be used to download a file through its Download API URL, provided that the HTTP client supports SSL and Basic Authentication). For example:

- Any browser can be used to download one file at a time by typing the download URL in the address bar. The user will be asked to enter their LabCAS username and password in the widget that pops up.
- CURL can also be used to download all URLs listed in a file (as generated through the Listing API), through a command like the following which will cycle through each line in the file:
 - `xargs -n 1 curl -O -g --netrc --verbose --location --remote-name --remote-header-name < urls.txt`

Listing API

The listing API transforms the API queries into list of URLs which can be used to download the files. A listing request is constructed from a query request by simply changing the word '**select?**' with the word '**list?**'. Note that listing requests will always return a list of files that belong to the collections or datasets that matched the constraints specified in the request. As always, results returned to the user will be limited to those files to which the user has access.

The following examples create list of downloadable files for all the queries specified previously:

- List files for all collections:
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/collections/list?q=*&wt=json&indent=true'`
- List files for all collections submitted by "Boston University":
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/collections/list?q=Institution%3A%22Boston+University%22&wt=json&indent=true'`
- List files for all datasets related to "Mammogram":
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/datasets/list?q=Mammogram&wt=json&indent=true'`
- List files belonging to a specific dataset:
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/files/list?q=DatasetId:Team_37_CTIIP_Animal_Models.CTIIP-2.0a&wt=json&indent=true'`
- List files in the previous dataset that were produced with a specific 'stain':
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/files/list?q=labcas.pathology%5C%3AStain%3APR&fq=DatasetId%3ATeam_37_CTIIP_Animal_Models.CTIIP-2.0a&wt=json&indent=true'`

Note that in all previous cases, a listing of all file URLs can be generated by redirecting the command standard output to a file, for example:

- `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/files/list?q=DatasetId:Team_37_CTIIP_Animal_Models.CTIIP-2.0a&wt=json&indent=true' > urls.txt`

Query API

LabCAS includes a query service that enables users to search for data of interest through a powerful query syntax. In fact, LabAS uses the same full query syntax as Solr, as it forwards the client request to the Solr server hidden behind the JPL firewall, adding the additional constraint to only return results to which the user is authorized.

The general syntax of a query request is: **`https://<host_name>/data-access-api/<object_type>/select?<query_constraints>`**

where:

- `<object_type>` = 'collections', 'datasets' or 'files'
- `<query_constraints>` = any query constraint support by Solr

Examples:

- Query for all collections:
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/collections/select?q=*&wt=json&indent=true'`
 - or using JWT:
 - `TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJMYWJDQVMiLCJleHAiOiE1NjE0ODExNjQsImh0bGU2MTQ4MDU2NH0.eyJhdWQiOiJMYWJDQVMiLCJleHAiOiE1NjE0ODExNjQsImh0bGU2MTQ4MDU2NH0.fZVZe_T50kuBQqIXYGHfCICJWUB_973RJGHQ2DbYxJY` (this is an expired token)
 - `curl -H "Authorization: Bearer $TOKEN" 'https://mcl-labcas.jpl.nasa.gov/data-access-api/collections/select?q=*&wt=json&indent=true'`
- Query for all collections submitted by "Boston University"
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/collections/select?q=Institution%3A%22Boston+University%22&wt=json&indent=true'`
 - note how the constraint: `Institution="Boston University"` must be quoted to match the exact string, and URL-encoded
- Query for all datasets related to "Mammogram":
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/datasets/select?q=Mammogram&wt=json&indent=true'`
- Query for all files belonging to a specific dataset:
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/files/select?q=DatasetId:Team_37_CTIIP_Animal_Models.CTIIP-2.0a&wt=json&indent=true'`
- Query for all files in the previous dataset that were produced with a specific 'stain':
 - `curl --netrc 'https://mcl-labcas.jpl.nasa.gov/data-access-api/files/select?q=labcas.pathology%5C%3AStain%3APR&fq=DatasetId:Team_37_CTIIP_Animal_Models.CTIIP-2.0a&wt=json&indent=true'`
 - note how the constraint key: `labcas.pathology:Stain` must be first encoded as: `labcas.pathology\Stain` because ':' is a special character in Solr used to separate key:value pairs in the query syntax; then the whole constraint: `q=labcas.pathology\Stain:APR&fq=DatasetId:Team_37_CTIIP_Animal_Models.CTIIP-2.0a` must be URL-encoded

Note that in all cases above, the URL can be typed directly in a browser window, resulting in the user being presented with a dialog to enter their LabCAS username and password.

User Data API

The LabCAS User Data API can be used to create/read/delete user session data (favorite collections, datasets, files, searches, etc.).

Security

All endpoints are subject to access control:

- The HTTP request must contain user credentials in the form of a LabCAS username/password pair, or a valid Jason Web Token that carries the user identity.
- Each user is only allowed to create/read/delete his/her own data - i.e. records with a unique identifier equal to their username

Schema

The user data metadata conforms to the following schema:

- id: the user login name, used as record unique identifier (string)
- FavoriteCollections: a list of collection ids (strings)
- FavoriteDatasets: a list of dataset ids (strings)
- FavoriteFiles: a list of file ids (strings)
- SavedSearches: a list of user saved searches (Strings). Each item has the form: <search_name>:<search_url>
- LastLogin: the date and time when the user last logged in, in ISO-8601 format and UTC time zone (YYYY-MM-DDThh:mm:ssZ)

Endpoints

All endpoints use JSON to encode input and output data. All HTTP parameters and JSON payload cannot contain unsafe characters ("`<`", "`>`", "`\"`", "`%`", "`*`", "`$`"), which will make the HTTP request to fail.

- **POST: /userdata/create**
 - Creates a new record using the supplied JSON payload
 - Example with username/password (read from ~/.netrc file):
 - `curl -Lkv --netrc -H "Content-Type: application/json" -X POST -d '{"id":"lcinquini", "FavoriteCollections":["abc", "xyz"], "LastLogin": "2019-10-30T12:00:00Z"}' -v 'https://labcas-dev.jpl.nasa.gov/data-access-api/userdata/create'`
 - Example with Jason Web Token:
 - `TOKEN=$(curl -Lkv --netrc 'https://labcas-dev.jpl.nasa.gov/data-access-api/auth')`
 - `curl -Lkv -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" -X POST -d '{"id":"lcinquini", "FavoriteCollections":["abc", "xyz"], "LastLogin": "2019-10-30T12:00:00Z"}' -v 'https://labcas-dev.jpl.nasa.gov/data-access-api/userdata/create'`
- **GET: /userdata/read?id=...**
 - Retrieves the user data for the specified user id
 - Example: `curl -Lkv --netrc 'https://labcas-dev.jpl.nasa.gov/data-access-api/userdata/read?id=lcinquini'`
- **POST: /userdata/delete?id=...**
 - Deletes the user data associated with the specified user identifier
 - Example: `curl -Lkv -X POST --netrc 'https://labcas-dev.jpl.nasa.gov/data-access-api/userdata/delete?id=lcinquini'`

LabCAS Developer Guide

This page provides information and links relevant to Laboratory Catalog and Archive Service (LabCAS)

- [LabCAS Back-End: Installation from Docker](#)
- [LabCAS Back-End: Installation from source](#)
- [LabCAS Back-End: Data Publishing](#)
- [LabCAS Back-End: NIST Workflow](#)
- [LabCAS Back-End: Changes](#)
- [LabCAS Project Questions](#)
- [LabCAS UI Use Cases](#)

LabCAS Back-End: Changes

This page documents the changes between evolving versions of the LabCAS Back-End

Changes from v04 to v0.5

- The "labcas-upload" workflow is now used to both create a new version of a dataset, or to update the data and metadata content of an existing version
 - if the metadata flag "--key newVersion true" is used, a new version of the dataset will be created
 - Otherwise, the same dataset version will be updated (and created from scratch if not existing already)
- The "labcas-update" workflow has been removed
- The first upload task configuration defines a larger set of metadata fields, as instructions to the UI that needs to display forms to the user:
 - All fields starting with "input.dataset.<field_name>..." refer to dataset (aka product type) metadata
 - All fields starting with "input.file.<field_name>..." refer to file (aka product) metadata
- Behind the scenes, LabCAS assigns semantically rich identifiers to all products, as opposed to opaque "UUIDs" as before. Identifiers are created according to the template structure: "<product type>.<version>.<product name>", for example: "id=mydata.1.file1.txt". This change should be transparent to the user, as product ids should NOT be parsed to extract any information.
- Much of the boiler-plate metadata that OODT automatically associates with each product has been removed. The product level metadata consists of only the core CAS fields, and the LabCAS/EDRN metadata fields.
- The Solr schema defines "DatasetId" and "Version" to be single-valued fields, while all other LabCAS/EDRN fields can have multiple values.

Changes from v05 to v0.6

- The "labcas-upload" workflow does not use the "labcas-upload" sub-directory to archive products. Instead:
 - If a product has the default parent product type ("EcasProduct"), then it is archived directly under \$LABCAS_ARCHIVE
 - If a product declares a non-default parent product type, such as "NistProduct", then it is archived under \$LABCAS_ARCHIVE/NistProduct
 - This change should be transparent to clients, except for the different download URLs
- The "nist" workflow has been completely re-factored to use the same functionality as the "labcas-upload" workflow, but it is configured with its own specific metadata.
- The "labcas-upload" and "nist" workflows has an additional required metadata field: "OwnerGroup"
- Metadata fields can be marked as ...visible=true (default) or ...visible=false (must be explicitly set), in which case the LabCAS UI does not need to display an input widget to the user, but simply set them
- Uploaded datasets can declare a specific "ParentDatasetId" which will cause:
 - Their product type to inherit the parent metadata elements
 - Their products to be archived in a sub-directory of the parent product directory

LabCAS Back-End: Data Publishing

Introduction

Large datasets are published to LabCAS by first uploading them to the LabCAS server, then invoking the LabCAS python client to send a publishing request to the OODT Workflow Manager. Mirror infrastructures are setup on the **labcas-dev.jpl.nasa.gov** and **labcas.jpl.nasa.gov** servers. It is strongly recommended that all publishing operations are tried first on the development server, then executed on the production server. All operations must be executed by the user "edrn":

- `ssh labcas-dev.jpl.nasa.gov` (or: `labcas.jpl.nasa.gov`)
- `ssh edrn@localhost`

A few useful environment variables are pre-defined:

- **LABCAS_HOME** = `/usr/local/labcas/backend/home` : root directory where the LabCAS software is installed
- **LABCAS_STAGING** = `/usr/local/labcas/backend/staging` : root directory where the data are staged before being published (unless the "--in-place" option is specified)
- **LABCAS_ARCHIVE** = `/usr/local/labcas/backebd/archive` : root directory where the data are permanently archived. For large datasets, collection sub-directories are symbolic links to other volumes.
- **LABCAS_SRC** = `/usr/local/labcas/src/labcas-backend` : git repository for LabCAS software
- **LABCAS_METADATA** = `/usr/local/labcas/src/labcas-metadata` : git repository for LabCAS metadata

Data Upload

Data can be published from two locations:

- From a staging area organized as: **\$LABCAS_STAGING/<collection_id>/<dataset_id>** (using the default workflow "labcas-upload"): in this case, during publication data are moved to the final location **\$LABCAS_ARCHIVE/<collection_id>/<dataset_id>/<version>**. This is the preferred option for medium size datasets.
- Directly from the final location **\$LABCAS_ARCHIVE/<collection_id>/<dataset_id>/<version>** (using the "--in-place" option which triggers invocation of the "labcas-upload2" workflow), which is typically a symlink to a large storage disk: in this case, data are not moved during publication. This is the preferred option for large datasets.

Collaborators will typically upload data to LabCAS using one of two options:

- SFTP: data end up on the server **zipper.jpl.nasa.gov** at the location: **/home/sftpxfer/data/<institution>**
- WebDAV: data end up on the **labcas** or **labcas-dev** server at the location **/data/dav/staging**

Before publishing can take place, data need to be manually moved by LabCAS staff from those upload locations to one of the two publishing locations listed above, and always organized in a hierarchy of type:<collection_id>/<dataset_id>.

Metadata

In addition to the data themselves, data providers must supply accompanying metadata at both the collection and dataset level. These metadata files are stored by LabCAS staff in the Git repository: <https://github.com/EDRN/labcas-metadata>. Typically, metadata are split between one collection-level file and many dataset-level files. The same collection metadata file can be used when publishing different datasets in that collection. The default behavior is to always update the collection metadata, unless the option "UpdateCollection=false" is specified, in which case only the dataset metadata will be updated.

Please refer to the document: [LabCAS User Guide](#) to determine which metadata fields are required and which are optional.

Data Publishing

The general syntax for publishing data into LabCAS is:

- **labcas_publish <paths to one or more metadata files> [--in-place] [--debug]**

where:

- alias `labcas_publish='python $LABCAS_SRC/common/src/main/python/gov/nasa/jpl/edrn/labcas/publish_labcas_data.py'`

The python client aggregates collection and dataset level metadata from all the metadata files specified as arguments, then sends a publishing request to the OODT Workflow Manager, and monitors its execution until completion. While the script executes, it is very useful to follow the server log `$LABCAS_HOME/cas-workflow/logs/cas_workflow.log` which will contain any error messages, for example caused by missing metadata fields.

Publishing examples:

- **MD Anderson Pancreas images** (located under `$LABCAS_STAGING/MD_Anderson_Pancreas_IPMN_images/IPMN_P1-06_H03`):
 - `cd $LABCAS_METADATA/MD_Anderson_Pancreas_IPMN_images`

- labcas_publish MD_Anderson_Pancreas_IPMN_images.cfg IPMN_P1-06_H03.cfg
- **RNA Sequencing data** (located under \$LABCAS_ARCHIVE/RNA_Sequencing/ERR164550):
 - cd \$LABCAS_METADATA/RNA_Sequencing
 - labcas_publish RNA_Sequencing.cfg ERR164550.cfg --in-place
- **NLST data** (located under \$LABCAS_ARCHIVE/NLST/218757, after the directory structured has already been flattened) :
 - cd \$LABCAS_METADATA/NLST
 - labcas_publish NLST.cfg 218757.cfg --in-place

At the end of a successful publication, it is always good a good idea to inspect the metadata indexed into the Solr catalog: <http://labcas-dev.jpl.nasa.gov:8983/solr/#/> (or: <https://labcas.jpl.nasa.gov:8983/solr/#/>) in the different cores: "collections", "datasets", "files", and to test browsing and download of data from the LabCAS front-end: <https://labcas-dev.jpl.nasa.gov/> or <https://labcas.jpl.nasa.gov/>.

Examples of Solr query URLs:

- collection metadata: http://labcas-dev.jpl.nasa.gov:8983/solr/collections/select?q=id:MD_Anderson_Pancreas_IPMN_images&wt=json&indent=true
- dataset metadata: http://labcas-dev.jpl.nasa.gov:8983/solr/datasets/select?q=CollectionId:MD_Anderson_Pancreas_IPMN_images&wt=json&indent=true or: http://labcas-dev.jpl.nasa.gov:8983/solr/datasets/select?q=id:MD_Anderson_Pancreas_IPMN_images.IPMN_P1-06_H03&wt=json&indent=true
- file metadata: http://labcas-dev.jpl.nasa.gov:8983/solr/files/select?q=DatasetId:MD_Anderson_Pancreas_IPMN_images.IPMN_P1-06_H03&wt=json&indent=true

Data UnPublishing

Data can be unpublished from labCAS by removing the corresponding metadata from the Solr index. Unpublishing operations will only delete the metadata, not the data themselves.

To unpublish a single dataset, and all its files:

- **labcas_unpublish_dataset <dataset_id>**

where:

- alias labcas_unpublish_dataset='/usr/local/labcas/scripts/delete_dataset.sh'

for example:

- labcas_unpublish_dataset MD_Anderson_Pancreas_IPMN_images.IPMN_P1-06_H03

To unpublish a whole collection, including all its datasets and their files:

- **labcas_unpublish_collection <collection_id>**

where:

- alias labcas_unpublish_collection='/usr/local/labcas/scripts/delete_collection.sh'

for example:

- labcas_unpublish_collection MD_Anderson_Pancreas_IPMN_images

LabCAS Back-End: Installation from Docker

This page explains how to install and run an instance of the LabCAS back-end services as a Docker container. Because of its simplicity, this is the recommended way to run LabCAS.

Pre-Requisites

You must have Docker Engine installed on your personal system (Mac OSX, Windows or Linux), a basic understanding of how Docker works, and preferably you should have run a test command to verify that your Docker installation does indeed work. For example, you can run the following command to verify that your Docker installation works:

```
$ docker run ubuntu /bin/echo hello world
```

References:

- Docker installation: <https://docs.docker.com/engine/installation/>
- Docker MacOSX tutorial: <https://docs.docker.com/docker-for-mac/>
- Docker Windows tutorial: <https://docs.docker.com/docker-for-windows/>
- Docker Linux tutorial: <https://docs.docker.com/engine/installation/linux/>

Installation and Operation

- Start a Docker terminal on your system
- Download the image:

```
$ docker pull edrn/labcas-backend:latest
```

- Start a container based on the previous image. Give the container a unique name ("mylabcas") and make the standard ports available to clients. The command will also create a terminal connection to the container, and log you in as the default user "labadmin" (which is a user who is authorized to run the LabCAS services):

```
$ docker run -p 8983:8983 -p 9001:9001 --name mylabcas -ti edrn/labcas-backend
```

- Once inside the container, start the LabCAS back-end services. Keep this terminal open to view the server-side logs.

```
[labadmin@3c283eb1b827]$ $LABCAS_HOME/start.sh
```

- Start another Docker terminal where you will be issuing client-side commands. Connect the second Docker terminal to the same running LabCAS container (by referencing the "mylabcas" name):

```
$ docker exec -it mylabcas /bin/bash
```

- Submit a test workflow:

```
[labadmin@3c283eb1b827]$ cd $LABCAS_HOME/cas-workflow/bin
[labadmin@3c283eb1b827]$ ./wmgr-client --url http://192.168.99.100:9001 --operation --sendEvent --eventName
labcas-test --metaData --key experiment 11 --key species snakes
```

- Verify that products were generated and ingested by using your browser to query the standard Solr URL. The IP address "192.168.99.100" is specific to the Docker Host you are currently running, and can be retrieved by issuing the command "docker-machine ip":

```
$ docker-machine ip
192.168.99.100
$ wget 'http://192.168.99.100:8983/solr/oodt-fm/select?q=%3A*&wt=json&indent=true'
```

- Stop the container (and the services inside the container). In the Docker terminal where you first started the container, type "exit". This will also disconnect the second terminal window.

```
[labadmin@3c283eb1b827]$ exit
```

- Restart the container. Connect again to that container via a shell, then restart the services. Verify that the published data (generated by the test workflow) is preserved across starting/stopping a container:

```
$ docker start mylabcas
$ docker exec -it mylabcas /bin/bash
[labadmin@3c283eb1b827]$ $LABCAS_HOME/start.sh
$ wget 'http://192.168.99.100:8983/solr/oodt-fm/select?q=%3A*&wt=json&indent=true'
```

- To create a new container from the same image, in a clean state (i.e. no data is restored), first delete the previous container, then restart a new one with the same name (or, create a new container with a different name). Then proceed as before to start all the services.

```
$ docker rm mylabcas
$ docker run -p 8983:8983 -p 9001:9001 --name mylabcas -ti edrn/labcas-backend
[labadmin@3c283eb1b827]$ $LABCAS_HOME/start.sh
```

- To start or connect to a container as "root" user, use the -u option to override the default image user:

```
$ docker run -p 8983:8983 -p 9001:9001 --name mylabcas -ti -u root edrn/labcas-backend
$ docker exec -it mylabcas -u root /bin/bash
```

LabCAS Back-End: Installation from source

LabCAS Version: v0.6

This page documents how to install the EDRN LabCAS Back-End software stack on a typical Unix server, starting from source (i.e. package by package). It was tested on CentOS, Amazon Linux, and MacOSX. The whole process should take less than an hour.

Pre-Requisites

Mandatory:

- Unix OS
- Java 1.7+ (JDK installation, not JRE)
- Maven 2.2+
- Git

Optional:

- Open port 8983 (within firewall, or to a specific IP only) to be able to access the Solr web interface
- Install solrpy to be able to query Solr through Python

Installation

- Define the environment where the software will be installed. Create the directories below if not existing already.

```
# location where all software components will be installed
export LABCAS_HOME=/usr/local/labcas/home

# root location where products will be uploaded (only needed to support publishing of uploaded files)
export LABCAS_STAGING=/usr/local/labcas/staging

# location where product files will be archived
export LABCAS_ARCHIVE=/usr/local/labcas/archive

# location where the specific executable will be installed (possibly outside of LabCAS version control)
export PGE_ROOT=/usr/local/labcas/backend/pges
```

- Check out the labcas-backend code from the repository (to any directory, for example ~/src)

```
git clone https://github.com/EDRN/labcas-backend.git
cd labcas-backend
# optionally select a specific branch - not recommended unless you are a LabCAS developer
# git checkout -b devel origin/devel
```

- Compile and install. Note that the first time you compile on a given server, it will take a longer time for Maven to download all the library dependencies. After the first time, compilation will go much faster.

```
# install all back-end services to $LABCAS_HOME
mvn install

# install specific workflows (as needed) on top of the existing services
mvn install -Dworkflow=labcas-test
mvn install -Dworkflow=labcas-upload
mvn install -Dworkflow=rnaseq-pipeline
```

Start/Stop Services

To start all back-end services (File Manager with [Tomcat/Solr](#) back-end, [Workflow](#) Manager, Resource Manager):

```
cd $LABCAS_HOME
./start.sh
```

To stop all services:

```
cd $LABCAS_HOME
./stop.sh
```

To start or stop a single service (for example):

```
source $LABCAS_HOME/env.sh
cd $LABCAS_HOME/cas-workflow/bin
./wmgr start
./wmgr stop
```

Example Command-Line Usage

To execute the labcas-test workflow:

```
cd $LABCAS_HOME/cas-workflow/bin
./wmgr-client --url http://localhost:9001 --operation --sendEvent --eventName labcas-test --metaData --key
experiment 11 --key species snakes
```

After the labcas-test workflow has completed:

- product-level metadata can be queried from the Solr catalog: <http://<hostname>:8983/solr/#/oodt-fm/query>
- job output and logs can be inspected at:
 - \$PGE_ROOT/labcas-test/output
 - \$PGE_ROOT/labcas-test/jobs
- product files will be archived under: \$LABCAS_ARCHIVE/labcas-test/files/

To trigger publishing of the dataset "mydata" previously stored in the \$LABCAS_STAGING/mydata directory:

```
cd $LABCAS_HOME/cas-workflow/bin
./wmgr-client --url http://localhost:9001 --operation --sendEvent --eventName labcas-upload --metaData --key
DatasetId mydata --key DatasetName 'My Data' --key Description 'My own data' --key ProtocolId 1 --key LeadPI
'John Doe' --key ProtocolName 'GSTP1 Methylation' --key DataCustodian 'Rich Smith' --key DataCustodianEmail
'rich.smith@pubmed.gov' --key CollaborativeGroup 'Prostate and Urologic' --key OwnerGroup EDNRN_CANCER_GROUP
```

To create a new version of the "mydata" dataset, with supposedly updated data/metadata content:

```
cd $LABCAS_HOME/cas-workflow/bin
./wmgr-client --url http://localhost:9001 --operation --sendEvent --eventName labcas-upload --metaData --key
DatasetId mydata --key DatasetName 'My Data' --key Description 'My own data' --key ProtocolId 1 --key LeadPI
'John Doe' --key ProtocolName 'GSTP1 Methylation' --key DataCustodian 'Rich Smith' --key DataCustodianEmail
'rich.smith@pubmed.gov' --key CollaborativeGroup 'Cancer' --key OwnerGroup EDNRN_CANCER_GROUP --key NewVersion
true
```

To query the OODT Workflow Manager for all available workflows:

```
cd $LABCAS_HOME/cas-workflow/bin
./wmgr-client -op -events -url http://localhost:9001
```

To query the Solr index (back-end to the OODT File manager):


```
# query all products
http://localhost:8983/solr/oodt-fm/select?q=%3A*&wt=json&indent=true

# query all files of type 'LabcasTestFile'
http://localhost:8983/solr/oodt-fm/select?q=CAS.ProductTypeName%3ALabcasTestFile&wt=json&indent=true
```

To download products (aka files) published to LabCAS, you can query the OODT Product Server deployed on port 8080 and supply the specific product unique identifier:

```
# generic wget syntax: wget --content-disposition 'http://localhost:8080/fmprod/data?productID=<product_id>'
wget --content-disposition 'http://localhost:8080/fmprod/data?productID=mydata.1.file1.txt'

# generic curl syntax: curl --remote-name --remote-header-name 'http://localhost:8080/fmprod/data?
productID=<product_id>'
curl --remote-name --remote-header-name 'http://localhost:8080/fmprod/data?productID=mydata.1.file1.txt'
```

To delete ALL records from the Solr index (use with great caution):

```
curl -s http://localhost:8983/solr/oodt-fm/update?commit=true -H "Content-Type:text/xml" --data-binary
'<delete><query>*:*/</query></delete>'
```

Example Python Clients

Several example Python clients are distributed together with the labcas-backend source code:

- [test_workflow_client.py](#): Python script that submits the "labcas-test" workflow
- [nist_workflow_client.py](#): Python script that submits the "labcas-nist" workflow
- [upload_workflow_client.py](#): Python script that submits the "labcas-upload" workflow

Upgrade

An existing LabCAS installation can be upgraded to the latest software release without losing any information about existing products and workflows.

To upgrade all components at once:

```
# fetch the latest labcas-backend code from the source repository
cd labcas-backend
git pull
cd labcas-backend

# will remove all existing software but NOT the existing data archive or metadata catalogs
# and install the core labcas-backend components
mvm clean install

# will install the latest version of the following workflows:
mvn install -Dworkflow=labcas-test
mvn install -Dworkflow=labcas-upload
mvn install -Dworkflow=nist
```

LabCAS Back-End: NIST Workflow

LabCAS Version: v0.6

Description

The LabCAS NIST workflow supports upload and processing of data files from distributed remote NIST laboratories. This workflow combines the functionality of the "LabcasUpload" workflow with some other NIST-specific tasks that process the data. Each lab uploads data to a specific directory (such as \$LABCAS_STAGING/NistLab01), creating its own LabCAS data type (such as "urn:edrn:NistLab01"), which is always a sub-type of "NistProduct". As usual in LabCAS, each NIST data type can have multiple versions. Once archived, the data files can be searched, downloaded, or used as input to the Shiny server.

Workflow

At this time the workflow is composed of the following tasks:

- **NistInitTask**: provides configuration information to the LabCAS UI, and upon execution it creates the specific product type and product type metadata (including a new version, if requested).
 - This is a standard "LabcasUploadInitTaskInstance" that is configured with NIST-specific metadata fields, as opposed to the default ECAS metadata fields.
 - The new product type creates is always a sub-type of "NistProduct"
- **NistConvertTask**: converts input data files into a common format, including adding some of the metadata provided in the workflow execution.
 - For now, it converts .txt and .csv files to .txt.nist, .csv.nist files (in the same directory)
 - For now, it embeds metadata as header lines
- **NistExecTask**: generates an application URL that can be used to analyze the data file
 - For now, an example application URL is: `http://shiny.jpl.nasa.gov/?datadir=/usr/local/labcas/backend/archive/NistProduct/NistLab01/1`
 - For now, the ApplicationURL metadata key, value are added to the file-level metadata stored in Solr
- **NistCrawlTask**: crawls the directory \$LABCAS_STAGING/\$DatasetId and publishes all the .txt, .csv and .nist files to the OODT File Manager
 - This is a standard "StdPGETaskInstance" configured to run the OODT Cawler with some specific client-side metadata extractors

Metadata

At this time, the following metadata fields (all required) need to be provided by the client wishing to execute a NIST workflow:

- **DatasetId**
- **LeadPI**
- **Instrument**
- **Lab**
- **Date**
- **ParentDatasetId="NistProduct"** (set automatically)

Invocation

Example of command line invocation using the OODT Workflow Manager client:

```
cd $LABCAS_HOME/cas-workflow/bin
./wmgr-client --url http://localhost:9001 --operation --sendEvent --eventName nist --metaData --key DatasetId NistLab01 --key LeadPI Johns --key Instrument trombone --key Lab Lab01 --key Date 20160101 --key OwnerGroup Lab01_OwnerGroup [--key NewVersion true]
```

The optional "NewVersion" parameter will trigger the generation of a new version for the NistLab01 dataset. Data products will be archived on the server in the location: \$LABCAS_ARCHIVE/NistProduct/NistLab01/<version>/

Example of Python script invocation:

```
import xmlrpclib

workflowManagerServerProxy = xmlrpclib.ServerProxy('http://localhost:9001/')
metadata = {'DatasetId': 'NistLab01',
            'LeadPI': 'Johns',
            'Instrument': 'trumpet',
            'Lab': 'Lab01',
            'Date': '20160101',
            'OwnerGroup': 'Lab01_OwnerGroup',
            'NewVersion': 'true' }
workflowInstId = workflowManagerServerProxy.workflowmgr.executeDynamicWorkflow( ['urn:edrn:NistInitTask', 'urn:edrn:NistConvertTask', 'urn:edrn:NistExecTask', 'urn:edrn:NistCrawlTask'], metadata )
```

LabCAS Project Questions

Questions for new projects/sites:

1. Describe the goal of this project:
 - a. What is the purpose of this project?
 - b. If data already accessible... is it about sharing it with a larger audience?
 - c. Or is the goal to make it searchable together with other collections?
 - d. Other?
2. Describe the data hierarchy:
 - a. What is a logical division of data into collection / datasets / files?
 - i. what is a good name for the whole data collection ?
 - ii. Will there be other collections similar to this?
 - iii. what identifies a dataset?
3. Describe the metadata:
 - a. Which metadata fields belong to which level?
 - b. Which are important for searching or should be captured in the repository?
4. What is the total size of the archive?
 - a. how many collections?
 - b. how many datasets?
 - c. how many images per dataset?
5. What are the access restrictions on the data?
6. How do we publish all the data ?
 - a. Do we transfer it to JPL then publish it?
7. What are the timelines/milestones?

LabCAS UI Use Cases

Use Cases

The LabCAS UI (and LabCAS back-end) must support the following use cases:

1. Upload data while creating a new Collection and new Dataset
2. Upload data into an existing Collection, creating a new Dataset
3. Upload data into an existing Collection and existing Dataset,
 - a. Changing the Dataset version number
 - b. Not changing the Dataset version number
4. Run a data processing workflow
 - a. Including uploading some data
 - b. Without uploading any data
5. Re-run a previous data processing workflow
6. Browsing of existing Collections, Datasets
 - a. Allow in-context upload ?

Releases

- **January 2017 Release:**
 - Support use case 2: uploading data into existing collections (Anirban/Jamie, Sandy, Avi)
 - Support use case 4a (NIST) and perhaps 4b

Implementation Details

Next release:

- Use case 2: use the workflow configuration metadata to guide the UI flow:
 - parameter "CollectionName"="NIST Product" (for example)
- Use case 4: again use configuration metadata to decide whether or not data upload is necessary:
 - parameter "UploadFiles"="true/false" (false by default ?)

Later releases:

- Use case 1:
 - start by clicking "Standard Labcas Dataset"
 - then show a combo box that allows to either choose an existing Collection, or enter the name of a new collection ?
- Use case 3:
 - use "NewVersion" flag = true/false

LabCAS on AWS

This section contains administrator level instructions for operating the LabCAS environment on AWS.

Currently, we support 3 instances of LabCAS running on AWS, each specific to a consortium, running on the same EC2 server (with internal IP "ip-172-16-1-172.us-west-2.compute.internal"):

- MCL
- EDRN
- CIB

Each LabCAS instance is deployed as a set of Docker containers that share a specific configuration, a set of data directories, and communicate through a shared Docker network. In each case, one of the containers is an Nginx server that proxies HTTP requests to the other containers, and that it is itself proxied by an AWS ELB to accept requests from the internet.

The following hostnames are defined:

- <https://labcas-mcl.jpl.nasa.gov/> <http://ip-172-16-1-172.us-west-2.compute.internal:80>
- <https://labcas-edrn.jpl.nasa.gov/> <http://ip-172-16-1-172.us-west-2.compute.internal:81>
- <https://labcas-cib.jpl.nasa.gov/> <http://ip-172-16-1-172.us-west-2.compute.internal:82>

[Developing with Docker LabCAS on your laptop](#)

[Start/stop AWS LabCAS](#)

[Publishing Data to AWS LabCAS](#)

[Using caMicroscope on AWS](#)

Developing with Docker LabCAS on your laptop

These instructions explain how to deploy Docker LabCAS onto your development environment, so that you can develop and test changes before deploying in production.

The LabCAS application is composed of 4 containers:

- **labcas-backend**: includes all the OODT back-end services: the File Manager, the Workflow Manager, the Product Server, and Solr.
- **labcas-client**: includes the client libraries for publishing/un-publishing data to the LabCAS back-end services. This container must have access to the same data directories as the labcas-backend.
- **labcas-ui**: includes the LabCAS user interface
- **nginx**: the front-end proxy that exposes selected LabCAS endpoints to the internet

The "labcas-..." Docker images are stored in the public "edrn" Docker repositories, and will be automatically retrieved when starting the application via Docker-compose. Nonetheless, when developing each image can also be built locally.

Environment

Before building or starting LabCAS, the following environment variables must be set:

- **LABCAS_PORT**: port used by the nginx container. When running multiple instances of LabCAS on the same server, they must use different values for **LABCAS_PORT**
- **LABCAS_CONSORTIUM**: a label that specifies the specific consortium this LabCAS instance is configured for
- **LABCAS_INSTALL**: root directory where all LabCAS components (back-end, front-end) will be installed
- **LABCAS_CERTS**: directory where the host certificate used to encrypt/decrypt cookies is stored

For example:

- export **LABCAS_PORT**=80
- export **LABCAS_CONSORTIUM**=mcl
- export **LABCAS_INSTALL**=/usr/local/labcas/mcl
- export **LABCAS_CERTS**=/usr/local/labcas/labcas_certs

Or more simply, for example:

- source ~/mcl_env.sh

Building

- **labcas-backend** and **labcas-client**:
 - git clone <https://github.com/EDRN/labcas-docker.git>
 - cd labcas-docker
 - docker-compose build
- **labcas-ui**:
 - git clone <https://github.com/EDRN/edrn.labcas.ui.git>
 - cd edrn.labcas.ui
 - docker build -t edrn/labcas-ui .

Configuration

The LabCAS backend and ui are configured through the following files, which must be placed under **\$LABCAS_INSTALL/config**:

- **labcas.properties**
- **ui.env**

A copy of these files for the different consortia can be found in the configs/ subdirectories on GitHub (minus the passwords). After copying these files to **\$LABCAS_INSTALL/config**, the LDAP password and the JWT secret key must be inserted in each of them.

Starting/Stopping

A LabCAS instance is started or stopped using docker-compose with the "project" flag equal to the specific consortium:

- cd <labcas-docker directory>
- docker-compose -p \${LABCAS_CONSORTIUM} up -d
- docker-compose -p \${LABCAS_CONSORTIUM} logs -f
- docker-compose -p \${LABCAS_CONSORTIUM} down

After starting, the LabCAS UI should be available at this URL:

- http://localhost:<LABCAS_PORT>

Pre-built Solr Index

By default, when LabCAS is first started, it will initialize an empty Solr index, which means there will be no collections/datasets/files to browse through the UI. It is possible to install a pre-built Solr index prior to starting LabCAS:

- git clone <https://github.com/EDRN/labcas-docker.git>
- cd labcas-docker
- rm -rf \$LABCAS_INSTALL/home/solr-index
- tar xvfz data/mcl/solr-index.tar.gz -C \$LABCAS_INSTALL/home/solr-index

Publishing Data to AWS LabCAS

Data Hierarchy

Data is stored in LabCAS in a directory referenced as \$LABCAS_ARCHIVE. Each data collection is organized as an arbitrarily deep directory tree, where:

- The top level directory represents the data collection:
 - \$LABCAS/<collection name>
- Any number of nested sub-directories are interpreted as nested datasets:
 - \$LABCAS/<collection name>/<dataset 1>/<dataset 2>/.../<dataset N>
- A dataset can contain one or more files:
 - \$LABCAS/<collection name>/<dataset 1>/<dataset 2>/.../<dataset N>/file1.txt, file2.txt, ...

S3 Storage

Data may also be replicated on S3 storage, for long-term preservation and exchange with collaborators. Currently, only one S3 bucket is setup:

- s3://nlstc-data

Metadata

Each collection or dataset directory must contain a metadata file that describes it, which *must* be named exactly as the directory name + extension ".cfg". This file is parsed at the time of publishing (on the client side) and its contents submitted as part of the publishing request. Examples:

- \$LABCAS_ARCHIVE/Team_37_CTIIP_Animal_Models/Team_37_CTIIP_Animal_Models.cfg (collection metadata)
- \$LABCAS_ARCHIVE/Team_37_CTIIP_Animal_Models/CTIIP-2.2/CTIIP-2.2.cfg (dataset metadata)

File-level metadata originates from 2 sources:

- Some metadata is automatically extracted from the files themselves, if they are of a type that can be parsed by standard libraries
- Additional metadata may be supplied in an XML file that is located in the same directory as the data file, with the same name + extension ".xmlmet"
 - Example:
 - EX15-0250-HE-EH.svs
 - EX15-0250-HE-EH.svs.xmlmet

Publishing

Publishing operations are executed by Python client scripts that submit XML/RPC requests to the OODT Workflow Manager, running on a specific port. The collection and dataset metadata contained in the .cfg files is transmitted as part of the request. The Python scripts can be executed directly from the AWS host, or from within a "labcas-client" container which has all the proper software installed, and that shares the same \$LABCAS_ARCHIVE volume as the "labcas-backend" container where the OODT Workflow Manager and other LabCAS services are running. The same data paths are used by both containers.

Before running any publishing operation, the correct environment must be sourced, for example:

- ssh edrn-aws-docker
 - Login with your JPL username and password
- ssh edrn_fn@localhost
- load the proper environment by sourcing one of the following files:
 - source ~/edrn_env.sh
 - source ~/mcl_env.sh
 - source ~/cib_env.sh

To publish all data, recursively, starting at the collection level:

- From the AWS host:
 - General syntax: labcas_publish_collection --collection_dir <collection directory> --solr_url=[http://localhost:\\$SOLR_PORT/solr](http://localhost:$SOLR_PORT/solr) --workflow_url=[http://localhost:\\$WMGR_PORT](http://localhost:$WMGR_PORT)
 - EDRN example: labcas_publish_collection --collection_dir \$LABCAS_INSTALL/archive/Automated_Quantitative_Measures_of_Breast_Density_Data --solr_url=[http://localhost:\\$SOLR_PORT/solr](http://localhost:$SOLR_PORT/solr) --workflow_url=[http://localhost:\\$WMGR_PORT](http://localhost:$WMGR_PORT)
 - MCL example: labcas_publish_collection --collection_dir \$LABCAS_INSTALL/archive/Team_37_CTIIP_Animal_Models --solr_url=[http://localhost:\\$SOLR_PORT/solr](http://localhost:$SOLR_PORT/solr) --workflow_url=[http://localhost:\\$WMGR_PORT](http://localhost:$WMGR_PORT)
 - CIB example: labcas_publish_collection --collection_dir \$LABCAS_INSTALL/archive/MAST --solr_url=[http://localhost:\\$SOLR_PORT/solr](http://localhost:$SOLR_PORT/solr) --workflow_url=[http://localhost:\\$WMGR_PORT](http://localhost:$WMGR_PORT)
- From the Docker container running on the AWS host:
 - docker exec -it <labcas client container name> labcas_publish_collection --collection_dir <full path to the collection directory INSIDE the containers> --solr_url <Solr URL INSIDE the labcas client container> --workflow_url <OODT workflow manager URL INSIDE the labcas client container>

- EDRN example: `docker exec -it edrn_labcas-client_1 labcas_publish_collection --collection_dir /usr/local/labcas/archive/Automated_Quantitative_Measures_of_Breast_Density_Data --solr_url=http://labcas-backend:8983/solr --workflow_url=http://labcas-backend:9001`
- MCL example: `docker exec -it mcl_labcas-client_1 labcas_publish_collection --collection_dir /usr/local/labcas/archive/Team_37_CTIIIP_Animal_Models --solr_url=http://labcas-backend:8983/solr --workflow_url=http://labcas-backend:9001`
- CIB example: `docker exec -it cib_labcas-client_1 labcas_publish_collection --collection_dir /usr/local/labcas/archive/MAST --solr_url=http://labcas-backend:8983/solr --workflow_url=http://labcas-backend:9001`

To publish a single dataset, without affecting the rest of the data collection:

- From the AWS host:
 - General syntax:
 - `labcas_publish_dataset --dataset_dir <full path to the dataset directory> --collection_name <name of the top level collection> --update_collection=False --solr_url=http://localhost:\$SOLR_PORT/solr --workflow_url=http://localhost:\$WMGR_PORT`
 - Example: `labcas_publish_dataset --dataset_dir $LABCAS_INSTALL/archive/Team_37_CTIIIP_Animal_Models/CTIIP-2.2 --collection_name "Team 37 CTIIP Animal Models" --solr_url=http://localhost:\$SOLR_PORT/solr --workflow_url=http://localhost:\$WMGR_PORT`
- From the Docker container running on the AWS host:
 - General syntax: `docker exec -it <labcas client container name> labcas_publish_dataset --dataset_dir <full path to the dataset directory INSIDE the container> --collection_name <name of the top level collection> --update_collection=False --solr_url <Solr URL INSIDE the labcas client container> --workflow_url <ODT workflow manager URL INSIDE the labcas client container>http://labcas-backend:9001`
 - Example: `docker exec -it mcl_labcas-client_1 labcas_publish_dataset --dataset_dir /usr/local/labcas/archive/Team_37_CTIIIP_Animal_Models/CTIIP-2.2 --collection_name "Team 37 CTIIP Animal Models" --update_collection=False --solr_url=http://labcas-backend:8983/solr --workflow_url http://labcas-backend:9001`

To update the collection metadata with new values read from the configuration file, without publishing any data:

- From the AWS host:
 - General syntax: `labcas_publish_collection --collection_dir <full path to the collection> --solr_url=http://localhost:\$SOLR_PORT/solr --update_datasets=False --update_files=False` (no workflow manager URL is needed since this request only involves metadata)
 - Example: `labcas_publish_collection --collection_dir $LABCAS_INSTALL/archive/Team_37_CTIIIP_Animal_Models --solr_url=http://localhost:\$SOLR_PORT/solr --update_datasets=False --update_files=False`
- From the Docker container running on the AWS host:
 - General syntax: `docker exec -it <labcas client container name> labcas_publish_collection --collection_dir <full path to the collection directory INSIDE the containers> --solr_url=<Solr URL INSIDE the labcas client container> --update_datasets=False --update_files=False`
 - Example: `docker exec -it mcl_labcas-client_1 labcas_publish_collection --collection_dir /usr/local/labcas/archive/Team_37_CTIIIP_Animal_Models --solr_url=http://labcas-backend:8983/solr --update_datasets=False --update_files=False`

To view the workflow manager logs as data is published:

- `docker exec -it <labcas client container name> tail -f /usr/local/labcas/home/cas-workflow/logs/cas_workflow.log`
- Example: `docker exec -it mcl_labcas-backend_1 cat /usr/local/labcas/home/cas-workflow/logs/cas_workflow.log`

Un-publishing

To un-publish a single dataset:

- From the AWS host:
 - `labcas_unpublish_dataset <dataset identifier>`
 - Example: `labcas_unpublish_dataset Team_37_CTIIIP_Animal_Models/CTIIP-2.2`
- From the Docker container running on the AWS host:
 - `docker exec -it <labcas client container name> labcas_unpublish_dataset <dataset identifier>`
 - Example: `docker exec -it mcl_labcas-client_1 labcas_unpublish_dataset Team_37_CTIIIP_Animal_Models/CTIIP-2.2`

To un-publish a whole data collection:

- From the AWS host:
 - `labcas_unpublish_collection <collection identifier>`
 - Example: `labcas_unpublish_collection Team_37_CTIIIP_Animal_Models`
- From the Docker container running on the AWS host:
 - `docker exec -it <labcas client container name> labcas_unpublish_collection <collection identifier>`
 - Example: `docker exec -it mcl_labcas-client_1 labcas_unpublish_collection Team_37_CTIIIP_Animal_Models`

Start/Stop AWS LabCAS

All instances of AWS LabCAS use the same Docker images, but are started using different environment settings to load specific configuration, and reference different data directories. When running docker-compose commands, the "-p" flag is used to reference a specific LabCAS instance aka a consortium.

- **Login to the AWS Docker server:**
 - Start an instance of the AWS workspace
 - Ssh to the AWS Docker server:
 - `ssh -i <your aws pem key> ec2-user@172.16.1.172`
 - Become root:
 - `sudo su -`
- **Setup the specific environment** (run one command only):
 - `source ~/mcl_env.sh`
 - `source ~/edrn_env.sh`
 - `source ~/cib_env.sh`
- **Start the specific LabCAS instance:**
 - `cd ~/src/labcas-docker`
 - `docker-compose -p $LABCAS_CONSORTIUM up -d`
 - `docker-compose -p $LABCAS_CONSORTIUM logs -f`
- **Stop the specific LabCAS instance:**
 - Setup the specific environment first (see above)
 - `cd ~/src/labcas-docker`
 - `docker-compose -p $LABCAS_CONSORTIUM down`

Additionally, to recreate the images locally:

- **Rebuild the images:**
 - Setup the specific environment first (see above)
 - `cd ~/src/labcas-docker`
 - `docker-compose build`
- **Push the images to docker hub:**
 - `docker-compose push`

Using caMicroscope on AWS

The caMicroscope server is deployed on the AWS LabCAS environment as a set of Docker containers. The same server instance is shared by all LabCAS environments: MCL, EDNR, CIB.

Start/Stop

- **ssh to the AWS Docker server and become root:**
 - `sudo su -`
- **Define the environment:**
 - `source ~/quip_env.sh`
- **To start caMicroscope:**
 - `~/scripts/start_quip.sh`
- **To stop caMicroscope:**
 - `~/scripts/stop_quip.sh`
- The top-level caMicroscope URL is:
 - <http://dev-temp-8000-795837955.us-west-2.elb.amazonaws.com/select.php>
- The URL that lists all images is:
 - <http://dev-temp-8000-795837955.us-west-2.elb.amazonaws.com/FlexTables/index.php>

Publishing data

Data is published to caMicroscope by running a script which lists all SVS files in a given directory, and passes this script to a program running inside the "quip-loader" container. Specifically:

- ssh to the AWS Docker server as user "ec2-user"
- Run the script passing the path to the dataset directory relative to the Docker mount for "/data/images":
 - Usage: `camicroscope_publish.sh <dataset path with respect to root dir>`
 - Example: `camicroscope_publish.sh mcl/archive/Team_37_CTIIP_Animal_Models/CTIIP-2.2`

LabCAS UI Help System

The following outlines the proposed help text users can access from within the LabCAS user interface. The "Help" link is accessible in the footer of the LabCAS UI web application and currently leads to a single page. Based on DMCC feedback we want to improve the help documentation to more accurately reflect the tasks users will want to accomplish with LabCAS.

Outline

1. Instances of LabCAS
 - a. EDRN, MCL, CIB
 - b. Obtaining additional JPL-hosted instances
2. Logging in
 - a. Username and password
 - b. Obtaining an account
 - c. If you forgot your password
3. My Collections
 - a. How data in LabCAS is organized
 - i. Collections
 - ii. Datasets
 - iii. Files
 - b. What data appears in "My Collections"
 - c. What is "All Public Collections"
4. Collection view
 - a. Metadata on a collection
 - b. Selecting a dataset
5. Dataset view
 - a. Metadata on a dataset
 - b. Files belonging to a dataset
 - i. Downloading a single file
 - ii. Downloading multiple files
 - iii. Handling large downloads
 - c. Selecting a file
6. File view
 - a. Metadata on a file
 - b. Downloading a single file from this view
 - c. Launching specialized viewing applications
7. Search
 - a. Entering free text to find
 - b. Navigating results
 - i. Tabs for matching collections, datasets, and files
 - ii. Facets for collaborative groups, disciplines, investigators, and organs
8. Uploading data into LabCAS
 - a. Using the Upload Files button
 - i. Selecting a kind of data
 1. Standard LabCAS dataset
 2. Other kinds of data
 - ii. Filling out the metadata form
 - iii. Dragging files into the browser or selecting files to upload
 - b. Handling large uploads
9. Open source
 - a. Github source code
 - b. Docker
 - c. Deploying your own installation of LabCAS

LabCAS User Guide

LabCAS ("Laboratory Catalog and Archive Services") is a web-enabled environment that allows users to publish, share, search and download a wide variety of biomedical datasets. This page contains information for end-users of the LabCAS portal and services.

Data Publishing

At present, data can be published into LabCAS in two possible ways: by using the LabCAS User Interface to upload and trigger ingestion of data; or by uploading data directly to the LabCAS server. Before publishing, data should be "curated" i.e. organized in a meaningful directory structure, and provided with enough metadata to make them useful to other users.

Data Structure

Data in LabCAS is organized according to the following logical hierarchy:

- **Collections:** broad sets of related data from the same study, the same analysis, or the same project.
- **Datasets:** different sets of related files within the same collection - for example, the different patients of a clinical study, or the different labs participating in a study.
- **Files:** all the files in a given dataset - for example, all the images for a single patient, or all the data supplied by a single lab.

Examples:

- Collection=RNA Sequencing, Dataset=ERR164773, Files=ERR164773_1.fastq, ERR164773_2.fastq, gene.counts
- Collection=University of Colorado Lung Images, Dataset=UCHSC_1467, Files=22021 P06-32 B4 Bronchus Intermedium x40 D1.jpg

Before publishing data into LabCAS, you should organize them in a meaningful directory structure conforming to the model above.

Metadata

In LabCAS, metadata can be associated to data at all 3 levels: collections, datasets, and files. When publishing, it is recommended that the following metadata fields be supplied to make the data searchable and more useful (mandatory fields are underlined). When using the LabCAS UI to publish data, some of these fields are automatically populated based on the user login, or the value of other supplied fields. [Draft MCL LabCAS metadata CDEs](#) or [Draft EDRN LabCAS metadata CDEs](#) provide detailed data element information.

Collection

- CollectionName: short name for high level data collection (example: MD Anderson Lung Images)
- CollectionDescription: a few sentences describing the high level data collection (ex: Lung images for clinical studies conducted during 2001-2010.)
- OwnerPrincipal (ex: uid=amos,dc=edrn,dc=jpl,dc=nasa,dc=gov - this is provided by the IC)
- Consortium (ex: MCL or EDRN)
- Discipline: (ex: RNA Sequencing, Pathology, Oncology, etc.)
- LeadPI (ex: Chris Amos)
- LeadPIId
- QASState (ex: Public)
- Organ (ex: Lung, Pancreas, etc.)
- OrganId (ex: 3)
- Institution (ex: Dartmouth, MD Anderson Cancer Research Center)
- InstitutionId
- ProtocolName
- ProtocolId
- CollaborativeGroup

Dataset

- DatasetName: short name for this dataset (ex: ERR318895)
- DatasetDescription: a few sentences describing this dataset (ex: Data for patient X)
- PubMedID
- Species
- Instrument
- SpecimenType
- DatasetURL (ex: http://someotherwebsite/patients?id=X)

File

- FileName: parsed from file system
- FileLocation: parsed from file system
- FileType
- FileSize: parsed from file system
- FileDescription
- ProcessingLevel

Any other custom metadata can be supplied and stored as well, at all levels of the data hierarchy.

Publishing via LabCAS UI

The LabCAS web portal (<http://labcas.jpl.nasa.gov>) provides a web-enabled workflow for uploading data to the server, and trigger publication to the LabCAS archive. The user is guided through the process of selecting the files from their own desktop, and populating the required and recommended metadata fields. Because uploading data through the browser is not very efficient, this process is recommended for datasets that are not very large (up to a few GB in size).

Uploading Data to the LabCAS Server

Very large datasets should be transferred to the LabCAS server via some means other than a web browser. At this time, LabCAS offers two ways to upload data directly: SFTP and WebDAV. Both protocols are supported on a variety of platforms including Linux, Mac OSX, and Windows. Before uploading data, please make sure to:

- Organize your data in a directory structure corresponding to one collection and one or more datasets, as described above
- For each dataset, provide as much metadata as possible in a file colocated with the dataset, and named <dataset>.cfg (see [example file](#))

Then proceed through one of the two methods below.

SFTP

In this section we describe how to upload data using a Unix-like system (macOS, Linux, FreeBSD, Solaris, etc.) and on Windows.

Unix

On Unix-like systems, do the following:

- Generate a public/private RSA key pair with the following command, or use an existing pair if you have one:
 - `ssh-keygen -t rsa`
- Send your public key (id_rsa.pub) to the LabCAS team so they can enter it in the list of allowed user keys
- Simply use the SFTP client of your choice to connect to the LabCAS server (with no username or password, since authentication is provided by the key), navigate to your assigned root directory, create new directories for the Collection and Dataset to upload, then transfer the data (all files in the current local directory, including the metadata file <dataset>.cfg):
 - `sftp sftpxfer@zipper.jpl.nasa.gov`
 - `cd data/<user directory>`
 - `mkdir <collection directory>`
 - `cd <collection directory>`
 - `mkdir <dataset directory>`
 - `cd <dataset directory>`
 - `put *`

Windows

On Windows systems, you first generate a public/private key pair:

1. Download and install both [PuTTY](#) and [WinSCP](#). Both are free, open-source software packages.
2. Installing PuTTY makes a new program available, PuTTYgen. Start this.
3. At the bottom, select SSH-2 RSA and enter 2048 bits. (This should be the default.)
4. Press "Generate".
5. Move the mouse around in the blank area to generate random numbers until the progress bar is filled.
6. Click "Save public key" and save the key to a file. Mail that file to JPL and we'll install it in the right place. Alternatively, you can click in the Public key box, select ALL the text, and paste that in an email. This key should be a single long line of text that starts with "ssh-rsa AAAA..."
7. Enter a passphrase (and confirm it) then press "Save private key". Choose a filename (say "LabCAS") and save it to your Documents folder.
8. Close PuTTYgen.

At this point, you'll need to await confirmation from JPL that your public key has been installed. Once you hear back, you can start uploading data with the following:

1. Open WinSCP.
2. Press the "Advanced" button.
3. On the left side, under "SSH", click "Authentication".
4. Under "Authentication parameters", click the ellipsis ... button under "Private key file".
5. Select the private key you saved in the Documents folder in step 7 (above). Click "OK".
6. Under "Session", select SFTP. For Host name, enter zipper-vm.jpl.nasa.gov; for user name, enter "sftpxfer".
7. You will be warned the first time you're connecting to an unknown server. Press the "Copy Key" button. Then press "Yes".
8. Enter the passphrase for the private key you created in step 7 (above). Check the box by "Remember password for this session". Then press "OK".
9. You now have a window on the left with your local files and on the right. To send a file, select it on the left and click "Upload". Or simply drag and drop. You'll be prompted the first time to set up transfer settings. The defaults are fine, so check "Do not show this dialog box again" and press "OK".

WebDav data upload

- Contact the LabCAS team to obtain a WebDAV username and password, which will be stored in the LabCAS LDAP database
- Use one of the supported clients on your desktop to upload the data (see below). The server URL is: <https://labcas-dev-dav.jpl.nasa.gov/staging/>. After logging in with your credentials, drop your data in the location staging/<collection>/<dataset>

- Mac OSX: macs come with a pre-installed WebDAV client. In the upper bar menu, choose Go > Connect to Server, and enter the URL above
- Linux: we recommend using the *cadaver* client (see <http://www.webdav.org/cadaver/>):
- Windows:

Note that at this time uploading data through SFTP or WebDAV will NOT automatically trigger data publishing (although it will in the near future). Please contact the LabCAS team to let them know new data is available, and to coordinate the publishing phase.

RDF Schema for LabCAS Data

The following proposes an RDF structure (not an official RDF Schema or RDFS) to serve as examples of what a potential RDF export from either LabCAS or the LabCAS UI might look like. This is demonstrative and serves to show how clients might be prepared to ingest LabCAS data.

Collections

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:labcas="http://edrn.nci.nih.gov/rdf/rdfs/labcas-1.0.0#"
  xmlns:edrn="http://edrn.nci.nih.gov/rdf/schema.rdf#"
  xmlns:edrntype="http://edrn.nci.nih.gov/rdf/types.rdf#"
  xmlns:dc="http://purl.org/dc/terms/">
  <labcas:Collection rdf:about="https://edrn.jpl.nasa.gov/labcas/collections/12345">
    <dc:Title>Autoantibody Biomarkers</dc:Title>
    <dc:Description>Lorem ipsum</dc:Description>
    <labcas:Organ>Lung</labcas:Organ>
    <labcas:Organ>Prostate</labcas:Organ>
    <labcas:Metadata>metadata here</labcas:Metadata>
    <labcas:Dataset rdf:resource='https://edrn.jpl.nasa.gov/labcas/datasets/12345-12' />
    <labcas:Dataset rdf:resource='https://edrn.jpl.nasa.gov/labcas/datasets/12345-13' />
    <labcas:Dataset rdf:resource='https://edrn.jpl.nasa.gov/labcas/datasets/12345-14' />
  </labcas:Collection>
  <labcas:Collection rdf:about="https://edrn.jpl.nasa.gov/labcas/collections/45678">
    <dc:Title>Barrett's Esophagus Methylation Profile Dataset</dc:Title>
  </labcas:Collection>
</rdf:RDF>
```

Datasets

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:labcas="http://edrn.nci.nih.gov/rdf/rdfs/labcas-1.0.0#"
  xmlns:edrn="http://edrn.nci.nih.gov/rdf/schema.rdf#"
  xmlns:edrntype="http://edrn.nci.nih.gov/rdf/types.rdf#"
  xmlns:dc="http://purl.org/dc/terms/">
  <labcas:Dataset rdf:about="https://edrn.jpl.nasa.gov/labcas/datasets/12345-12">
    <dc:Title>Autoantibody Biomarkers</dc:Title>
    <dc:Description>Lorem ipsum</dc:Description>
    <labcas:Parent rdf:resource='https://edrn.jpl.nasa.gov/labcas/datasets/12345-10' />
    <labcas:Child rdf:resource='https://edrn.jpl.nasa.gov/labcas/datasets/12345-21' />
    <labcas:Child rdf:resource='https://edrn.jpl.nasa.gov/labcas/datasets/12345-22' />
    <labcas:File rdf:resource='https://edrn.jpl.nasa.gov/labcas/files/12345-12-1' />
    <labcas:File rdf:resource='https://edrn.jpl.nasa.gov/labcas/files/12345-12-2' />
    <labcas:File rdf:resource='https://edrn.jpl.nasa.gov/labcas/files/12345-12-3' />
  </labcas:Dataset>
  <labcas:Dataset rdf:about="https://edrn.jpl.nasa.gov/labcas/datasets/45678">
    <dc:Title>Barrett's Esophagus Methylation Profile Dataset</dc:Title>
  </labcas:Dataset>
</rdf:RDF>
```

Files

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:labcas="http://edrn.nci.nih.gov/rdf/rdfs/labcas-1.0.0#"
  xmlns:edrn="http://edrn.nci.nih.gov/rdf/schema.rdf#"
  xmlns:edrntype="http://edrn.nci.nih.gov/rdf/types.rdf#"
  xmlns:dc="http://purl.org/dc/terms/">
  <labcas:File rdf:about="https://edrn.jpl.nasa.gov/labcas/files/12345-12-1">
    <dc:Title>(By Batch)14-3-3.pdf</dc:Title>
    <dc:Description>Lorem ipsum</dc:Description>
    <dc:Type>application/pdf</dc:Type>
    <labcas:Size>4567124</labcas:Size>
    <labcas:FileID>long hex string here</labcas:FileID>
    <labcas:Directory>/tmp/dir</labcas:Directory>
    <labcas:Thumbnail rdf:resource='https://edrn.jpl.nasa.gov/labcas/a56b35e3.png' />
    <labcas:Dataset rdf:resource='https://edrn.jpl.nasa.gov/labcas/datasets/12345-12' />
  </labcas:File>
  <labcas:File rdf:about="https://edrn.jpl.nasa.gov/labcas/Files/45678-10-1">
    <dc:Title>Jin_Table1_LogisticRegression.doc</dc:Title>
  </labcas:File>
</rdf:RDF>
```