Labstory

Integrated ELN and LIMS

Users Guide

Table of Contents

LICENSE

If you have bought support, refer to your separate special license agreement.

This is an alpha release. Unless you have received special permission, you are not allowed to use it to spread it further nor modify it nor study it for the purpose of commercial competition.

Use is on own risk. There is no promise that future versions of the software will be able to read documents produced in this version. In particular, it might eat your dog.

Introduction

Labstory is an integrated Electronic Lab Notebook (ELN) and Laboratory Information Management System (LIMS). It is designed to meet patentability requirements as well as good lab practice (GLP) requirements. This means that it must be possible to trace documents and prove their origin. At the same time, it borrows the idea of distributed editing from the next-generation version control system (VCS), which have developed independently from ELN with heavy emphasis on supporting large teams. This means that the need of a central authority is removed; in particular, there is no need to be continuously connected to a server, and documents can be shared freely without the boundary of your organization. If a friend across the atlantic wants to have a copy of your protocol, he can have your unmodified unconverted lab journal page, and he can cross-reference it without thinking of which server it is located on (each document has a world-unique ID).

On top of this core design, we have tried to make everything available that one could only dream of in the days of paper journals – recording video, taking camera pictures straight into the pages, consistent book-keeping with databases, you name it. Either from your shiny new Apple mac, from your geeky Linux workstation, or your hip Android phone. We proudly claim that you are now holding one of the most sophisticated documentation systems in your hands and we wish you a happy time using it!

Digital signing

Basic principles

A common complaint about paper is that it is easy to forge documents. Digital files are by default much easier to forge – it is impossible to check them for DNA or age. However, with the advent of e-commerce and trade, techniques for secure transfer and protection against forgery have been developed. Digital signatures are used to protect digital documents from forgery and, if correctly used, provides much higher security than any paper signatures can provide.

A signature relies on having two cryptographic keys: one which is private (and is never given out), and one that is public. Anyone with the public key can verify that the signature was made by a person using the private key. The signature could be done as this:

1. Alice with the private key generates a text such as "...this is my document...and this is the date of the signature"
2. Alice encrypts the text with her private key
3. Bob receives the text and decrypts it using the public key
4. Because only Alice could encrypt it, Bob knows the document was signed by Alice

This software uses a variant that in addition relies on using a hash function (call it H). The hash function generates a fingerprint H(document) which is much smaller than the original document. The point here is that if someone else signs the document, the entire encrypted document need not be sent back but only a smaller version, speeding up transfer and reducing disk usage. The only caveat is that the function H must be a "secure hash function", that is, it must be difficult to find a forged copy such that H(document)=H(forged document). Luckily, good such functions exist.

Counter signing

A problem occurs when you cannot trust the one signing the document – Alice can at any time generate a document, with any creation date of her choice. This is a big problem if you wish to apply for a patent, and claim that she did it before her competition. The solution to this problem is counter-signing: Bob can sign not only her document, but also her signature. If Bobs signature can be trusted, then it proves that Alice signature is authentic and was not produced at a date later than that of Bobs signature (assuming Bob did his homework checking this, which the software does automatically). So, Bob can now join Alice to the patent court and witness in her favor. The more people counter-sign the document, the more people can help her witness.

What if Alice and Bob work at the same company. Are they not biased, having similar interests? This software helps you let other trusted people outside your organization counter-sign your documents.

TODO

Implemented signing methods

Several hash functions are supported and new methods can be used as they become available. The default hash function is currently SHA-512.

By default the software supports the use of PGP cryptographic keys (IETF RFC 4880). As such, these can be shared by other software, in particular for signing and/or encrypting email. Options for exporting/importing these keys can be found in the User profile menu.

The software is prepared to support the X.509 cryptographic standard. This is the standard used by authorities issuing for example bank cards, and would allow signing using personal smart cards. Other signing methods can also be used in tandem. Contact us for further details.

Storing the private keys on a server

This software provides the option of storing private keys on a central server. This is a handy feature because some users just can't seem to keep track of their files... However, it comes with security considerations. What could the server do with the keys in worst case?

It is a hard requirement that a server administrator *can not* use the keys stored on the server. Imagine if the administrator could sign document with one user, and counter-sign with all other users of the system – the administrator might be biased toward helping the own institute or a particular user. This would mean that all users of this server have to be treated as one in court, and only external users could be trusted for counter-signing! In addition, the keys could be leaked to the outside (central key deposits are prime targets for hackers) and be used for ID-theft.

Thus this software is designed to make it impossible for the administrator to use the keys. All keys are password protected, and the password is never sent to the server. Rather, the keys are downloaded to the client on demand. That said, with enough computational recourses, a given key can be unlocked, so it is still an utmost responsibility for the server administrator to not let the keys end up in the wrong hands.

The issue of storing keys on the server should be considered when comparing to other competing ELN systems. A system without personal cryptographic keys is equivalent to a system where the administrator has full access to all the keys. Thus these systems have serious concerns about the credibility of signatures. Even if they do have personal cryptographic keys, it is important to check that the password is not transmitted to the server.

Cryptography and transfer security

If nothing is done, a program sends the content of a data transfer as raw data. This is for example the case of HTTP. Unfortunately, this data can be read by other users having access to the network, allowing them to catch passwords and read the contents of the documents. In fact, it is even possible to hijack a connection.

This software uses state of the art encryption standards to ensure that nothing of the transfer is leaked to third parties nor that they can hijack the connection. This is done by encapsulating the transfer using the SSL protocol (IETF RFC 5246). Thus no extra measures need to be taken by the user or administrator to secure the traffic.

Version control

Keeping track of changes in a document is easy as long as only one user is editing the document, on a single computer. In this case it would suffice to store a dated copy of the document once in a while. However, what if you give the document to two colleagues to allow them to proof-read it? You will receive two altered copies back – three if you worked on it in the meanwhile. How do you get all the changes back into a single document? One solution is to simply only allow one person to edit the document at a time, but this can sometimes be unproductive. This is a problem programmers have faced on a large scale for many years and over time they have developed a solution to it that allows everyone to edit the document simultaneously: Version control. This software provides the latest developments in version control programmers and provides it for lab journal editing.

Version control requires that each copy of the document are given a formal version number. How this works will be illustrated by an example:

User Alice creates a new document and starts writing it. Once her working day is over, she sends it out for others to edit it if they wish. The document history at this point simply looks like this:

(1)*

This document is viewed by several people, but only Bob and Charlie decide to edit it. It now looks like this:

(2)* ← (1) → (3)*

(should one show the edit tree on each computer?)

Bob sends his changes back. In this case Alice can easily just step up to his latest version, because it is the latest available for her. For Alice it looks like this:

(2)* ← (1)

Now Charlie sends his copy, and since it does not stem from the latest from Alices point of view (2), it means that it contains conflicting changes and these conflicts must be resolved. Alice takes the changes in (2) and (3) and merges them, and produces a new document:

(2) ← (1) → (3)

→ (4)* ←

The new document (4) joins the two versions, and restores there being only one latest copy.

Some terminology from version control should be introduced here:

- The act of assigning an edited document a version is called *committing*.
- A version is called a *branch* (as on a tree)
- Joining branches is also called *merging*, and handling the differing changes is called *conflict resolution.*
- When merging only requires picking the latest version (as in going from 1 to 2), it is called a *fast forward*.
- The final branches in a version history are called *heads*. There is always at least one head, and if there are several, they must at some point be merged.

Much of the work is done by the software. Before you send a document to someone else, the latest version of the document must have been given a version, that is, you will be asked to *commit* it. As long as you receive versions which are direct descendants of the previous one, *fast forwards* will be done automatically. The only time you will need to take action is whenever documents must be merged.

The software does it's best to merge automatically but you must always approve the new version, and sometimes help with the merge (e.g. if two authors disagree on how a piece of text should be revised, you have to pick which way). However, as long as changes are not done at the same place, the software should be able to figure out how to best merge the documents.

The permission system

Labstory has a flexible system for controlling access to documents. It may be customized for local installations for most likely it will be similar to what it presented here.

The permissions are based on a Role-Based Access Model (RBAC). It is based on the following principles:

- One role has permissions that are system wide (e.g. administrator privileges)
- One role has permissions for one particular document (e.g. read access)
- One role can inherit privileges of another role (the latter role is often used as a group)
- **TODO**: One role may have have ownership of another role (e.g. owner of a group)

All roles are implicitly inheriting the permissions of the role u:guest. Thus anyone who is able to log in, by any means, will never have less privileges than the guest.

Obtaining a role on login

When logging in, the server assigns you roles depending on how you authenticated and the server configuration. You may for instance log in using your private cryptographic key, which commonly will assign you the role a:*YourAuthorID*. If you login by user/pass, you may obtain u:*UserName*. However, additional roles may be assigned depending on local fine tuning.

Permissions for roles

There are only a few permissions available for roles:

a	Administrator (can do anything)
c	Can create new documents

Unless you are an administrator, chance is that they will be already be set up for you and you will never have to think about them.

Permissions on documents

Permissions on a document apply to one particular role and the following ones are default:

v	Document is visible
r	Can read the document
e	Can edit the document (limited)
E	Can edit the document (full access)
d	Can delete document
s	Can share permissions with others

When a document is first created, some set of standard permissions will be applied to it. It is expected that the roles of the one that created it will at least get “vres”-access. This can be set by the administrator. After creating it, the user may wish to add additional permissions to let his/her group be able to read or edit it.

The limited edit access makes it illegal to modify signed parts of the document or delete signatures. Thus once a part of the document has been signed, the editing is locked for all future. New signatures may however be added at any time point, though with less value than had the document been signed right away.

LIMS - Laboratory Information Managements Systems

Documents are good for keeping track of day to day tasks in the lab. Their free structure allows any activity to be properly documented. But if you keep doing the same kind of task over and over, you might want to be able to properly organize the data such that it easily can be summarized. To do so you need to enforce a higher level of structure on your data, taking the first step toward setting up a laboratory information management system (LIMS). Labstory has all the facilities needed to set up both very small and very large structured databases, linked to ELN records.

Database concepts

To simplify life, disregard that you are working with a document-oriented ELN system. Instead, think that want to set up a set of cross-linked tables. Two simple tables are given below:

Strains		
Strain name	**Species**	**Description**
ce-1	C.elegans	Worms. These are green
ce-2	C.elegans	Worms. These are red

Freezer			
Freezer rack	**Tube position**	**Frozen by**	**Strain name**
5	A7	Johan	ce-1
3	J3	Niklas	ce-1

The first table shows which strains have been worked on in the lab. The second table shows what exists in the freezer. The first concept to introduce is that of relations and keys. Shortly, the two tables can be described as:

Strains:

Strain name → Species, Description

Freezer:

Freezer rack, Tube position → Frozen by, Strain name

Strain name REFERENCES Strains(Strain Name)

In the terminology we will be using here, columns are also called “Attributes”. The other concepts you need to know are “Relations”, “Primary keys” and “Foreign keys”.

The rows above including the attributes are called “Relations”, the first key concept. The importance of the relation is that it explains what information can be derived from the information you have. For example, the first relation tells you that if you know the Strain name, you can look up which species it is, and any further description of it. The second relation shows that you need to know both the Freezer rack and Tube position to tell what is located there – which strain and who froze it. The information on

the left side of an arrow, what is needed to derive all information, is called the “primary key” of the relation.

The second table has an attribute that references the first table. This is called a “Foreign key”. What this says is that by knowing the freezer position, it is also possible to derive all the information in the first table by combining them. That is, when combining the relations, you get:

Freezer rack, Tube position → Frozen by, Strain name, Species, Description

To make a relationship truly useful there must be constraints on the content. The keys now encountered are the first and most important constraint. In a table there cannot be two rows having the same value of the key. For the first table it means that there cannot be two strains with the same name. For the second table it means that there cannot be two tubes at the same location. The foreign key imposes constraints between two table as the value of foreign key has to match any value in the referenced table. Thus in the example it is not possible to freeze a strain that does not exist in the first table.

If you have followed thus far then you know the fundamentals of a modern relational database!

Attributes: Types and constraints

Another fundamental type of constraint is what values a certain attribute can have. Is it a text, or is it a number? If it is a text, can it be on multiple lines? For numbers, maybe they cannot be negative. All of this is encoded in the “Type” of an attribute. The following types are supported:

- Texts
- Integer numbers
- Decimal numbers
- Boolean values (True/False)
- Time points
- Files (strictly speaking, references to files **)

In addition, for each type it is possible to tell if it can be NULL. Such a record means that the information is missing (N/A). By default this is not allowed, but sometimes it is useful.

An attribute is referenced by the short name. This name will primarily be used by programmers accessing the data, using for example R, who benefit from less typing. For all other users, the long name will be displayed. Since the long name is only used for display you may modify it at any time without affecting the stored data. The short name, on the other hand, should preferably never be changed. So pick it wisely. In addition to the names, there is a description of the field. It is only used to present users with a help text when they are entering the data, and has no other function.

** To be able to retrieve the data of a file, you must have access to the original ELN record inserting the file

Designing a new database

Creating a new database is simple following these steps:

1. Start by writing down all the attributes that will be involved

2. Figure out the relationships. Break them into pieces such that there is no redundancy. In the example, the two tables could have been written as one, but then for example the description field would have to be duplicated every time a strain is frozen. This is really bad design. If you wish, you can look up “third normal form”/3NF, which is preferred in databases.
3. Create the tables. See if the design makes sense

Modifying an existing database

There will be a “duh” moment when you realize that you forgot something in the design of your database. This need not be a problem – if you for example just add a new attribute, a new foreign key, a new constraint, change a default value of an attribute etc, chance is Labstory will just cope with this. Old inserted entries will have a null value of missing attributes. If null is not allowed according to your design, the default value will be used instead. If instead an attribute is dropped, the attribute will just be ignored in old entries.

Relational databases vs ELN – how it works in Labstory

The tricky part comes when the relational database viewpoint meets the ELN viewpoint. In Labstory it works the following way:

- A table is created in a document. The document ID and position within the document is used as the name for the table. This guarantees that the table ID is world-unique (or as unique as the document ID at least).
- Insertions to a table, or deletions, are all done in ELN documents through special objects.
- Labstory then takes all documents, finds the declarations of tables, and all modifications, and assemble them into a final populated table. This table can then be viewed separately from the documents.
- Further, the assembled table can be accessed from external programs, and various statistics can be collected using SQL expressions.

Thus the data moves from an ELN point of view to a relational point of view, after processing of the software.

SQL database integration

Labstory can use a database in two different ways:

- An existing external table can be viewed and cross-referenced after it is first imported as a special Labstory table. Labstory will never modify it
- Labstory tables are assembled in an SQL database. By pointing Labstory to use a database of your choice, it is possible to cross-reference Labstory tables externally. But they cannot be modified, as Labstory may (or will) overwrite changes at any moment.

Labstory can interact with any JDBC-compliant database, that is, almost all common database implementations.

Using the Linux/Mac/Windows client

Systems requirements

The requirements depend heavily on how you use the software: If you keep local copies of all your documents, if you create videos, and if import large documents. Ranges include:

- 200Mb - 1Gb disk space
- 100Mb - 300Mb free RAM (beyond what is used by the operating system itself)

As for the operating system,

- Windows XP and above
- OS X 10.5 or later, Intel CPUs
- Any modern Linux, but preferably Debian or Ubuntu

Installation

Installation is very straight-forward. You may download a ZIP-file for your operating system of choice. Unzip it and place the content wherever you prefer. The location does not matter.

Windows

You need to install the Java JRE (or JDK) to be able to run the software. You can obtain it from here: http://www.oracle.com/technetwork/java/javase/downloads/index.html

Ubuntu/Debian

If you use Ubuntu or Debian there is a package file you may use. These consume less disk space as they share libraries with other software in your system. *Warning: The ZIP-files are more reliable*.

Debian (continue with Ubuntu instructions afterwards):

```
sudo apt-get install python-software-properties
```

Ubuntu:

```
apt-add-repository 'deb http://ppa.launchpad.net/qtjambi-community/libqtjambi/ubuntu maverick main'
apt-get update
apt-get install libqtjambi
```

Location of data files

If you need to move the data files from one system to another, we generally recommend using the export facilities. However, if you wish to try a manual move of the files, they are stored in different locations depending on the operating system.

Unix: $XDG_CONFIG_HOME/labstory/ (if not set, .config/labstory/)

Mac: ~/Library/Application support/labstory/

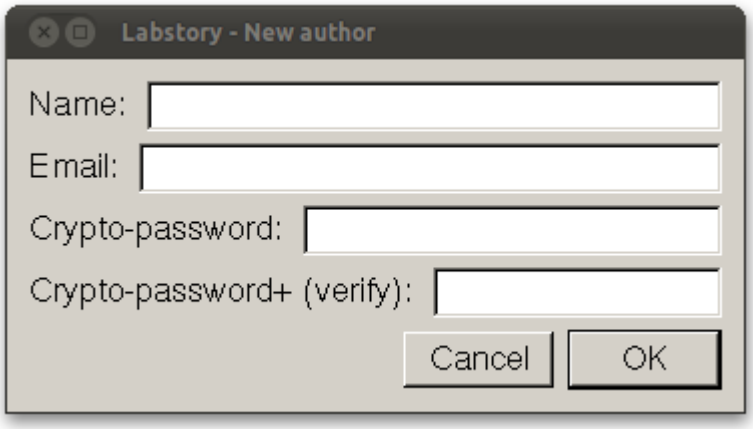
Windows: C:\config\labstory\

Working with authors

Before being able to write any documents, a user profile must be created and selected. If you have used the system previously you should import the previous profile to keep using the same cryptographic key.

Creating a new user profile

A new user profile is created by Client → User → Manage profiles →New profile. You will be asked to create a new cryptographic key and to set a password for it.



Using PGP keys for encrypting E-mail

If you decide to use PGP for signing documents then you have the option of also using them for signing and encrypting E-mail. This way you only use one “ID card” for both purposes, and thus also the same web of trust (see introduction for a discussion of this). If you have not used PGP before then you can export the keys and import them into your E-mail program.

TODO write how

If you already use PGP for E-mail, you can instead import these keys into your profile. First export them from your E-mail client (see the documentation for the software). Then, create a new profile in Labstory. Finally use “import” to import the keys, overwriting the previous keys of the profile.

TODO write more

Author favourite attributes

To sort your documents later, you will be setting attributes such as which group and project they belong to. Before doing so, you need to set up a list of favourite attributes (the system could just show all attributes that are on the server but the list would be too long to be convenient). You can store the favourites on a server to synchronize it between different clients. In addition, other users can optionally access it – this makes it easy to share the most commonly used attributes in your research group.

To see the attributes, go to ….....

TODO

Here it is also possible to create new attributes.

TODO maybe not allow to get others favourites. But could send them to someone via notifications. Server can use public key to verify that someone only downloads their own favourites

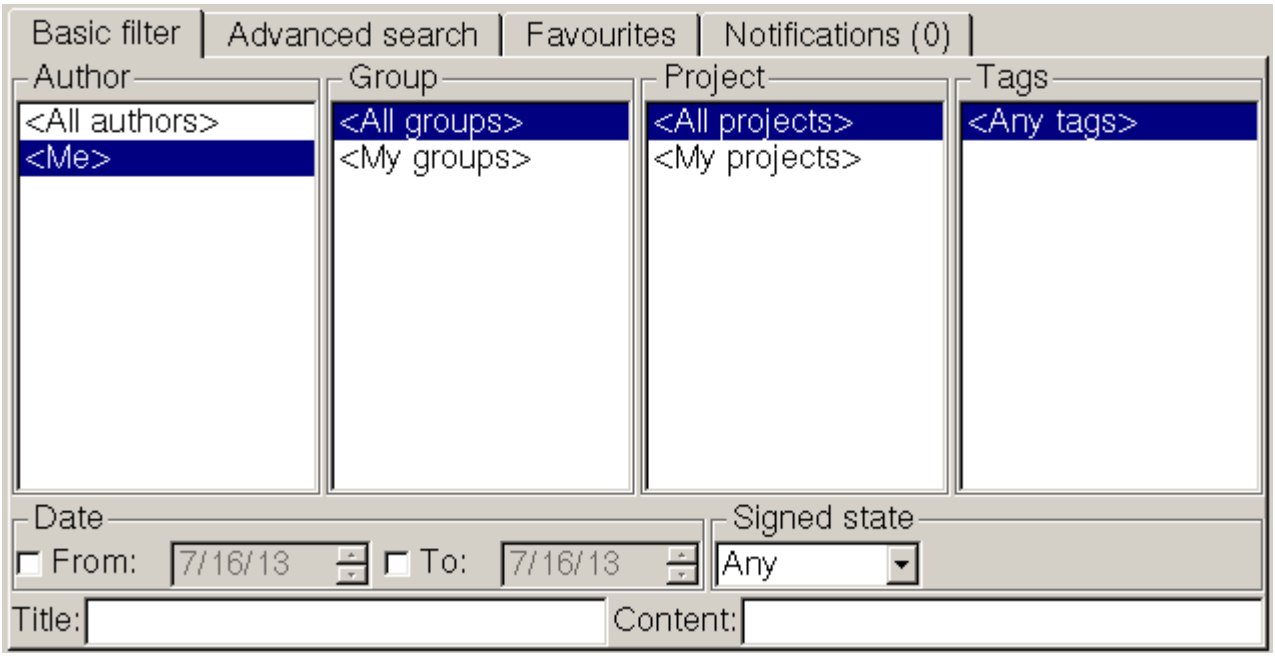
Uploading/Downloading a user profile from a server

Importing/Exporting a user profile

Searching documents

Basic filtering

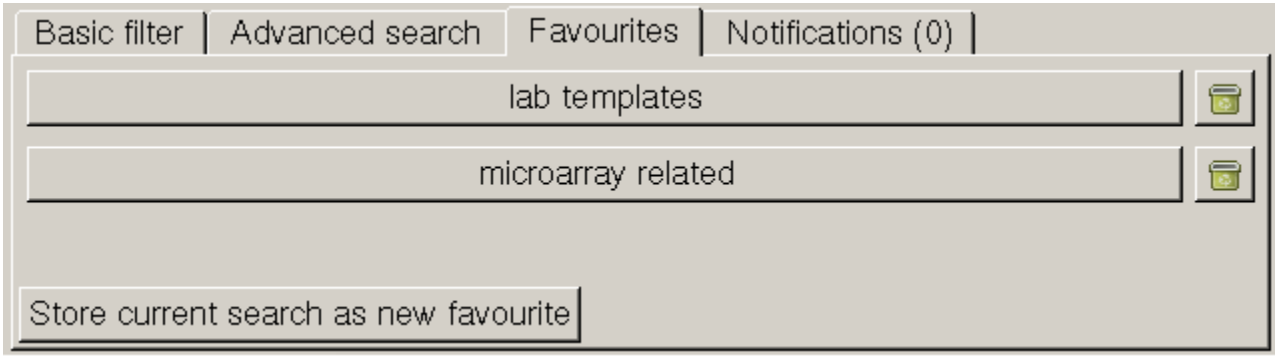
The basic search allows you to find documents that match all the given criteria simultaneously. You may impose filter by Author, Group, Project, Tags, Date range, Signed state and Title. In addition, you can do full-text search (Content).



Since the system may contain a very large list of entities, and to speed up the selection process, there are pre-defined groups such as *Me* (you current profile), *My groups* (any groups your current profile is listed in) and so on. To edit these, go to your author profile setting and configure the attributes (see section: Configuring author attributes).

Storing favourite searches

It is possible to store your most common searches as Favourites.



To do so, perform a search, using for example Basic filter. Then go the the favourites pane and click “Store current search as new favourite”. You will be asked to give it a name.

To recall a query, press the button. If you want to see the query you can hold the mouse cursor over the button for a few seconds. The trash can deletes the favourite.

Advanced searching

If the basic filtering is too rough, it is possible to instead search for documents using boolean queries. For example,

Author:”Carl” AND Group:”Meuwitz”

will search for documents with Carl as an author, and the document belonging to the Meuwitz group. Similarly, it is possible to use OR. To do the opposite, see what documents where generated while not in the Meuwitz group, the query would be

Author:”Carl” AND NOT Group:”Meuwitz”

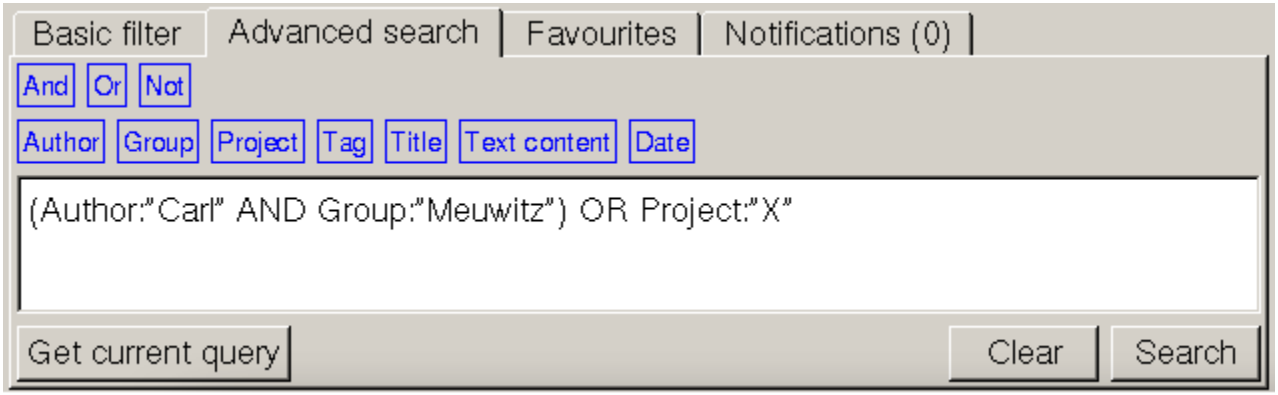
Whenever there is an ambiguity in the search pattern, it is possible to use ()-brackets. For example,

(Author:”Carl” AND Group:”Meuwitz”) OR Project:”X”

and

Author:”Carl” AND (Group:”Meuwitz” OR Project:”X”)

do not mean the same thing. The first will list any document from project X, while the latter will always require Carl to be an author.



Keywords are always of the form *Type:Value*, where *Type* is case-insensitive and for example any of Author, Group, Project etc. You don't need to remember the key words as they can be automatically typed by pressing the corresponding button.

If you wish to improve upon a filter you have created using for example a basic filter, you can use the “Get current query” button to insert the current search as a boolean query.

Working with documents

Creating new documents

You may either choose to create a blank document using “New document”, or you may use a template. Using a template is equivalent to copying a previous document and giving it a new ID and author. To do so, select a template document in the list and press “New from template”.

To create a new document you must have an author profile selected.

Filling in the document header

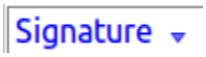
At the top of the document is the header. Some of it is already filled in when you create a document, e.g. the time of creation and the current author.

The most important field to fill is the title, or Labstory will not allow you to save your document. In addition, you may add attributes such as the Group and Project the document belongs to. Before you can do this, you need to set up the favourites for your author profile (See chapter on working with authors).

Working with document sections

A document consists of a number of sections, each of a particular type. The most important type is the Text section, but then there is the Graphics section to store images, and so on. To create a new content, press “Add content” and select the type of section you want.

It is possible to remove or rearrange sections. To do so, you can bring a menu in the upper-left corner of each section:



You may move a section up or down, one step at a time. Repeat as many times as needed. Note that the order of sections can not be altered after signing (coming sections).

Signing and witnessing

Once you consider the document to be in a correct and complete state, you should sign it. To do so, create a signature document section, and press Sign. You will be asked to choose an author and enter your password. After signing, the sections the signature refers to may no longer be modified.

Note: You can delete a signature, but with high likelihood the server will not accept that you overwrite a previous document with one that no longer has the signature. However, you can safely do so until you have uploaded your document and/or made a revision. Also note that any tags, projects, etc, will be locked by the signature.

Next, after signing it, you should have it witnessed by someone else to increase the trust of your signature (see the general section on witnessing). This is simply accomplished by having someone else sign your document, after you have signed it. Their signature will lock your signature from future modifications.

Errata

If you find out that you have made a mistake, and already signed the document, you may not simply modify the previous text. However, you can continue adding content after the signature. Simply create another text section and explain what was done wrong. Then sign and witness the document again.

Document sections

Text sections

File attachment

A file attachment section consists of one or more uploaded files. This section is the one to use if there is no more specific section for your data, e.g. videos and images.

There is no limit on file size *per se*, but your server administrator might have imposed local restrictions. Thus see your local recommendations. However, certain types of raw data are better stored other ways which provide more specialized access. For example, raw microscopy images can be stored in OMERO or PACS servers. Raw sequencing data (FASTQ, SAM, BAM) should be delta-compressed (not zipped) before they are small enough for final archiving. The tools for this are evolving rapidly and not listed here.

Graphics

TODO

Sound and video attachments

It is possible to attach video and sound files. A multitude of codecs are available for compressing multimedia files. To ensure that the files can be played with ease in the future, only a limited number of codecs are supported.

Video codecs: H.263 and H.264 AVC

Audio codecs: OGG (MP3 has been excluded for patent reasons)

LIMS sections

These are described in their own chapter on structured content management.

Notifications and messages

To be able to communicate with other users of the software, you are able to send them notifications. Notifications are stored on the server until a client connects and synchronizes. Thus notifications can be read and deleted off-line, but to send new notifications the client has to be online.

Asking for witnesses and attention

The single most important feature is being able to ask people counter-sign your documents. You can also just ask them to have a look at a document (for whatever the purpose might be). To do so, first ensure you are connected to a server. Then select a number of documents in the view. Finally:

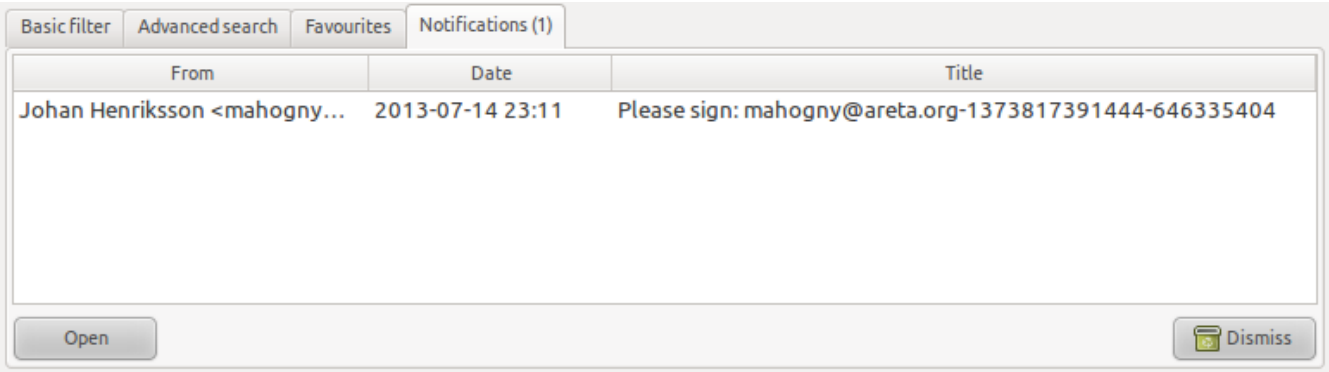
Menu: Document → Ask for signature

or

Menu: Document → Ask for attention

Reading notifications

To read notifications, look at the *Notifications*-tab. You can immediately see how many notifications you have within the parentheses (...).



To see which document a notification refers to, you can select one or more notifications and they will act as filters.

You can open a notification by double-clicking it or selecting it and pressing *Open*. If you have for example a signature request, this causes the relevant document to be opened.

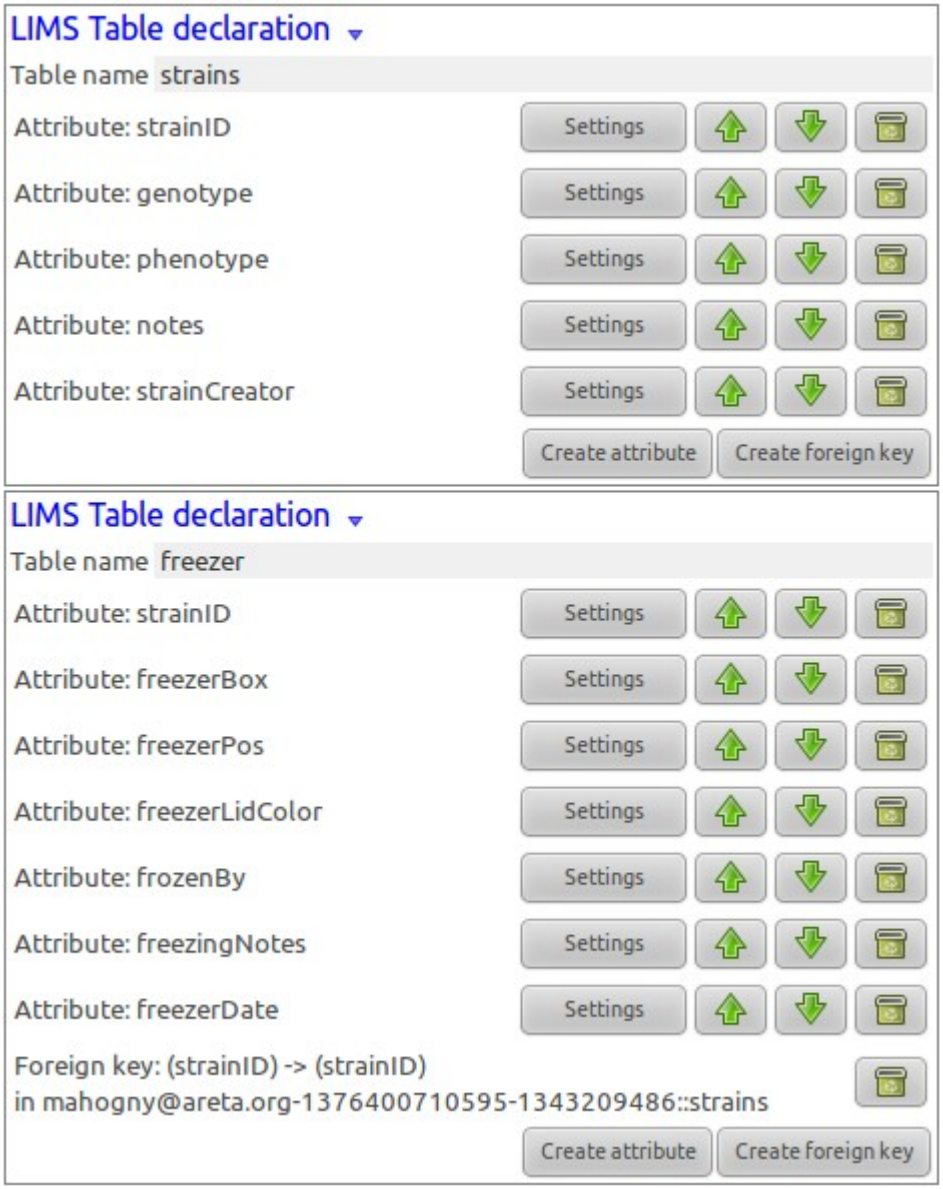
Once you consider a notification taken care of, select it, and press *Dismiss*.

Structured content management – LIMS

This chapter explains how to use the LIMS functionality. You are assumed to already know how to create new documents, and have read the LIMS theory chapter.

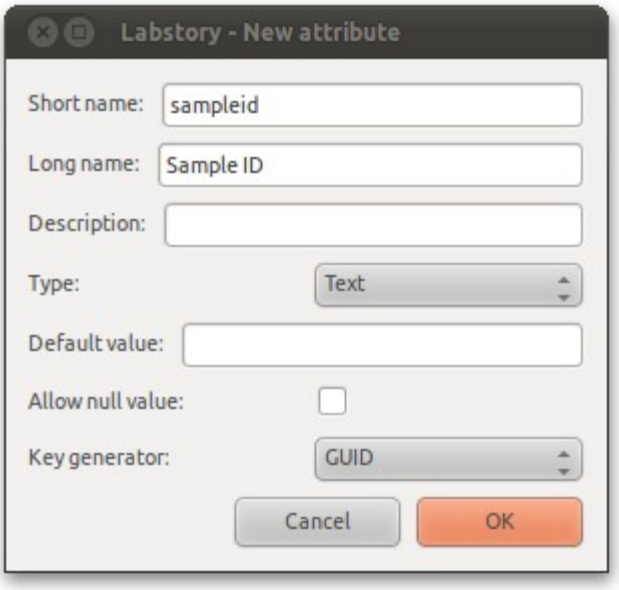
Creating new tables

New tables are created in a new document, using the “LIMS Table declaration” section. Below two tables are created, one for strain definitions and one for keeping track of frozen tubes of the strain:



One table is created at a time and the name is mandatory. In case you have signed your document and cannot modify the table record, it is possible to override the old one by creating a new declaration at the end with the same name.

When you create a new attribute you will get the following window. You are strongly discouraged from changing the short name once you have created the table and inserted values. Likewise also the type, but all other fields are decently safe to edit at any moment.

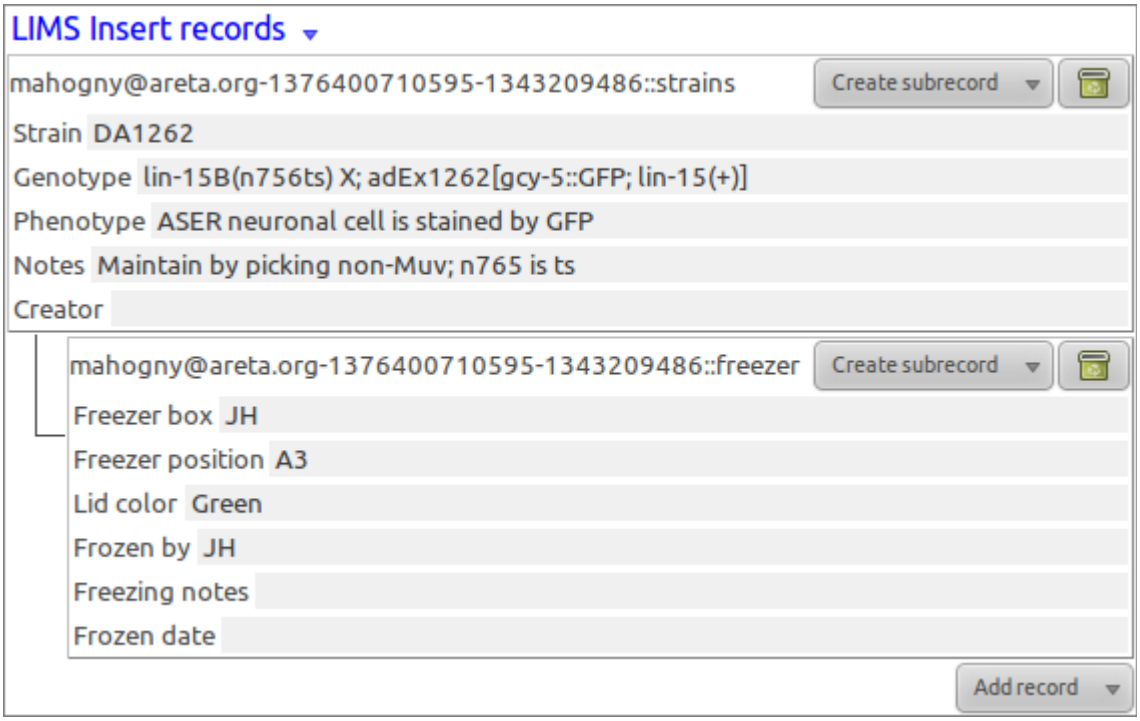


If you want to cross-reference another table (again, see the general theory chapter), you can create foreign keys. Pressing the “Add foreign key” button, you will see the following window:

On the top you select which table to reference, below the attributes to include. You need to include at least one attribute to create a key. The window will try to automatically match up attributes having the same name, thus consistent naming will make you more productive.

Inserting values

To insert values, create a document containing “LIMS Insert records” sections. You are discouraged from creating a table and inserting values into it, in the same document. One insertion section with two records is shown below:

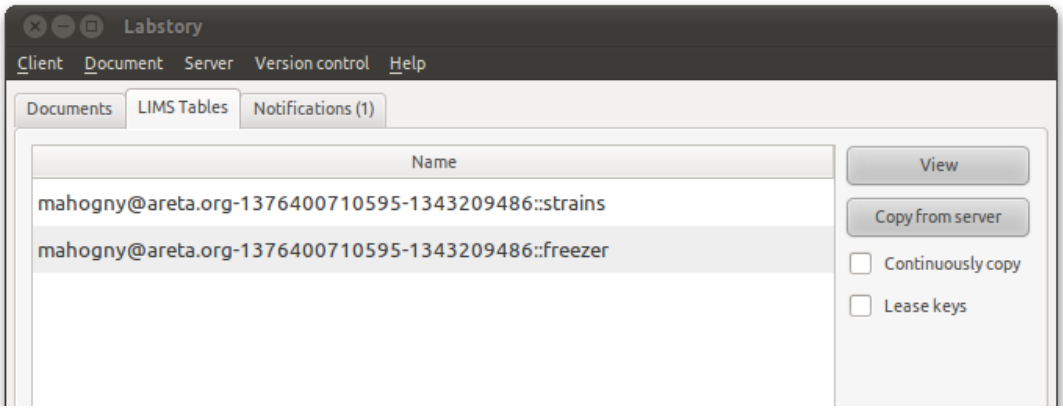


You can insert multiple records into multiple databases, but for clarity you are recommended to make one section for each type of database. In particular this enables you display more compact tables of multiple insertions (which is impossible if you mix record types). Click “Add record” to add more records.

In addition, it will use the foreign keys to let you create a hierarchy of related entries immediately. For example, if you enter a sample record, you can immediately also add records where it has been frozen, or how it has been further processed. To do this, click “Create subrecord” and a list of suggested referencing tables will be shown. By doing subrecords immediately you also ensure that the cross-referencing is done correctly as those values for subrecords are filled in automatically.

Managing tables

Once you have documents defining tables and inserting data, they will be indexed by the system. To display which tables you have, in go the “LIMS Tables” tab in the main window. You will get to the following view:

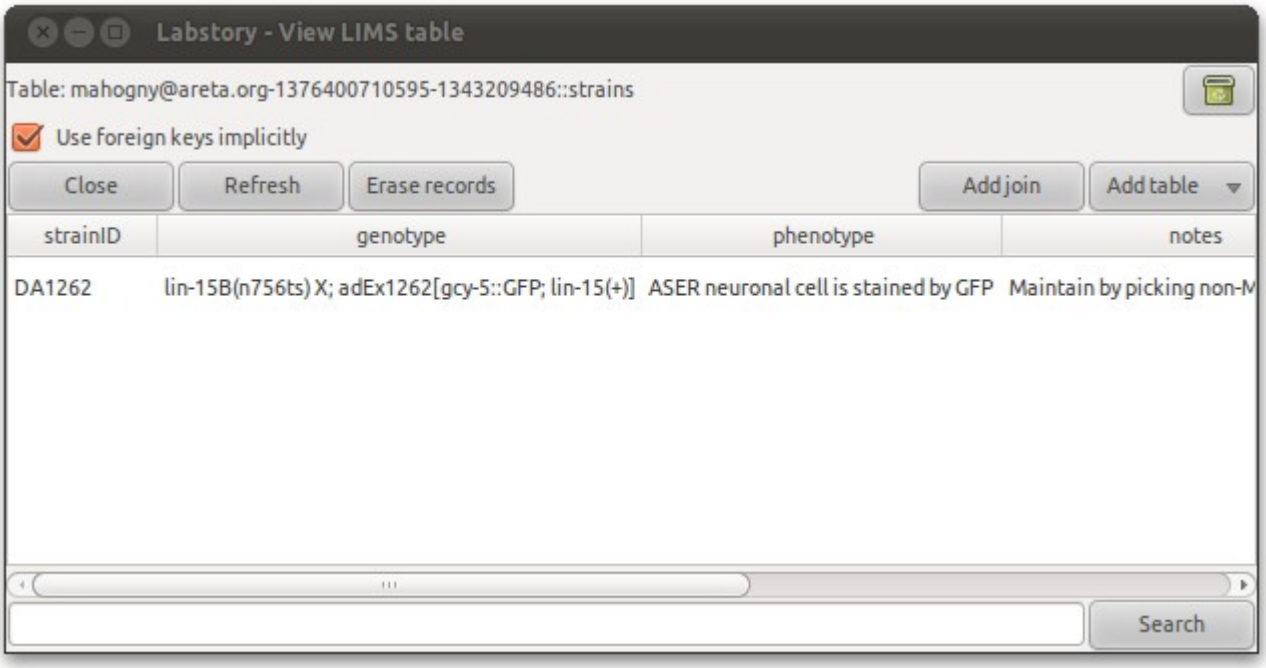


“View” will open a display of the table contents. The other options are for managing off-line use of the table.

- “Copy from server” will create a copy of the table in your local client. Thus, you need not download all the documents to keep the table content when browsing off-line.
- “Continuously copy” means to have the client synchronize with the server every time it connects. This operation is costly (CPU and bandwidth) and thus you must explicitly enable it for the tables you are interested in
- “Lease keys” is used for tables with a narrow key space. Tables set to generate GUID keys do not need key leasing, but for example, the table might be set to generate small integers as ID:s. In such cases, to be able to generate ID:s off-line you need to tell the server that you are reserving a set of ID:s for future use. Since you should not lease keys for tables you will never edit, you have to explicitly enable leasing for your tables. **TODO** write more about this.

Viewing tables

If you double-click a table in the table list, you will open a display of the content:



If the result table is too long it may be truncated, in which case you may wish to filter the result, or you can explicitly load a larger part of the table.

In the view you can also show merged (joined) tables. Use “Add table” to include more data. However, you will need to tell how the tables should be merged. By default, “Use foreign keys implicitly” is checked, meaning that any foreign keys you have declared will be used to match up the rows. In case however wish to cross-reference your data with another table it was not originally meant to reference (e.g. data from another research group), you can specify new merging rules using “Add join”.

TODO support viewing like input records

TODO support tree views

TODO export tables, csv and XML

Deleting and modifying values

Deleting values is more complicated than inserting values, and it is encouraged to design tables in such a way that a deletion is never necessary. To modify, you need to delete the old record and then insert a new record in it's place. Thus tables requiring modifications are also discouraged.

To delete a value, you will actually be undoing the insertion. Thus you need to find a reference to the insertion first. Start by viewing the table (or a merge of tables). Select cells from the table you wish to delete. One cell per entry is enough – if you only display a single unmerged table then you can use any column. Then press “Erase records”. This will not immediately erase the records but you have to create a new document that does it. So create a document or continue with one you are editing. Then use the

button “Paste new section”. This will cause a new delete section to be inserted. Once the document is saved the delete will be executed.

Searching with SQL expressions

Filtering is done using SQL boolean queries. For example, you execute:

```
sampleid LIKE “%ce%” AND strainSource=”CGC”
```

This is a pure boolean query, similar to filtering documents using the advanced search. However, SQL allows you to search with cross-referencing, e.g.:

```
sampleid LIKE “%ce%” AND NOT (sampleid in SELECT sampleid from “..... :: frozen”)
```

Here pulling our samples containing “ce” in the name, and which are not yet frozen.

The Android client

System requirements

- The client is designed for Android 2.2
- It is primarily tested with a Samsung Galaxy Note
- 100Mb-500Mb space

Installation

Download the file to your device, then use for instance *Apk installer* (https://play.google.com/store/apps/details?id=com.graphilos.apkinst&hl=en) to unpack it.

General

The Android client is designed as an addition, not a replacement, for the PC client. Thus this document assumes familiarity with the latter and will mainly highlight differences.

Android devices have several limitations which make it difficult to run large pieces of software on them. Primarily, the speed and amount of memory limit how large documents can be handled. If you know you will be using this platform a lot, try to adjust your workflow to not create too large documents but rather several smaller ones. Since the platform is not meant to entirely replace the PC platform, some features might also not be accessible, either because they consume too much memory or because they don't fit the smaller screen well. Combine the platforms for the most efficient use of your Android device.

The Android devices however also come with new possibilities – in particular the built-in camera allows you to use media for your documentation in ways you have likely not done before.

APPENDIX: STANDARDS

The following are links to some relevant standards:

CFR 21 FOOD AND DRUGS, part 11 - ELECTRONIC RECORDS; ELECTRONIC SIGNATURES
http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfCFR/CFRSearch.cfm?CFRPart=11&showFR=1

CFR 37 INTELLECTUAL PROPERTY REGULATIONS
http://www.ecfr.gov/cgi-bin/text-idx?c=ecfr&tpl=/ecfrbrowse/Title37/37tab_02.tpl

OpenPGP Message Format (IETF RFC 4880)
http://tools.ietf.org/html/rfc4880

The Transport Layer Security (TLS) Protocol version 1.2 (IETF RFC 5246)
http://tools.ietf.org/html/rfc5246